Causal Type System for Ambient Movements

Torben Amtoft
Department of Computing and Information Sciences
Kansas State University
tamtoft@cis.ksu.edu
www.cis.ksu.edu/~tamtoft

Abstract

The Ambient Calculus was developed by Cardelli and Gordon as a formal framework to study issues of mobility and migrant code. We present a type system for the calculus, parameterized by a set of security constraints: static ones concerning where a given ambient may reside, and dynamic ones expressing where a given ambient may be dissolved. A subject reduction property then guarantees that a well-typed process never violates these constraints; additionally it ensures that communicating subprocesses agree on their "topic of conversation".

The type system employs a notion of causality in that processes are assigned "behaviors". We argue that this significantly increases the precision of the analysis and compensates for the lack of "co-capabilities" (an otherwise increasingly popular extension to the ambient calculus); also it allows (in contrast to other approaches) an ambient to hold multiple topics of conversation.

Based on techniques borrowed from finite automata theory, type checking of type-annotated processes is decidable. Under certain quite natural restrictions, type inference is also possible.

1 Introduction

The ambient calculus was developed by Cardelli & Gordon [CG98] as a framework for mobile computation where "ambients", containing active processes (and not just passive code), can move around—in and out of other ambients. In this view, the following question becomes a main security concern:

is it possible for a given ambient to end up inside another given ambient?

Various approaches have been taken to analyze the ambient calculus. First came the type system of [CG99], where the focus was on ensuring that within each ambient there is a well-defined "topic of conversation"; later this system was extended [CGG99] so as to express also mobility properties. To give an (approximate) answer to the question above, many tools have been employed: flow analysis [NNHJ99], abstract interpretation [HJNN99, LM01], tree grammars [NN01], 3-valued logic [NNS00].

In this paper we address the issue using a type system where a process is assigned a so-called "behavior". A behavior[1] incorporates causality information, thus enabling one to express that one

[1]The concept dates back at least to [NN94] where it was used in the context of Concurrent ML, the type system of which has a very different flavor from the one considered in this paper.

action precedes another. In [AKPG01] this notion was used to extend [CG99] so as to allow *consecutive* topics of conversation (akin to session types for the π-calculus [GH99]). Since behaviors can be viewed as finite automata, well-known techniques can be used for type checking and inference.

The original ambient calculus is quite liberal in that ambients may enter, exit, and even open (in effect dissolve), other ambients without their explicit permission. In particular the opening capability is problematic for security[2], as seen by the reduction rule

$$\mathsf{open}\ n.P \mid n[Q] \rightarrow P \mid Q$$

which shows that the process Q inside the opened ambient n will run on the same level, and therefore with the same control power, as the opening process P. Also for developing a precise analysis, the opening capability is problematic, as we essentially need to split the work of Q into two parts: what is done before n was opened (not influencing P), and what is done after n is opened (influencing P). As pointed out in [CGG99], conservatively assuming that all actions of Q *might* take place *after* n is opened prevents us from declaring immobile an ambient that opens an entering (hence mobile) ambient.

A solution, originating[3] in [LS00] and employed in [DCS00, AKPG01, GYY01], is to add an extra "co-open" construct enabling an ambient to specify exactly at which point of its computation it will allow itself to be dissolved.

It is possible, however, to stick with the original calculus, and still get a quite precise analysis. For that purpose, it is essential to keep track of how the location of an ambient changes over time, thus allowing an estimate concerning when it can be opened. This information can (cf. above) be achieved by a variety of methods [NN01, NNS00, LM01]; the main contribution of this paper is to show how it can be achieved using the technology of type systems and finite automata. A detailed comparison with other systems is left for future work.

Our type system is (implicitly) parameterized with respect to a set of security constraints, listing which interactions between ambients are allowed. One view is to assume that these constraints are prescriptive; establishing that a process is well-typed thus verifies that no other interactions may happen. Alternatively, we might view the constraints as descriptive; a type inference algorithm might then deduce the least set of constraints needed for typability.

As in [CGG00] we shall employ the notion of *groups*, with the intention that each ambient belongs to exactly one group $g \in \mathsf{Grps}$ (a finite set). We shall use G to range over sets of groups. *Dynamic* security constraints are then expressed using the predicate

$$\mathcal{O}(g_0, g)$$

saying that an ambient of group g_0 is allowed to be opened while directly enclosed in an ambient of group g, whereas *static* security constraints are expressed using the predicate

$$\mathcal{I}(g_0, g)$$

saying that an ambient of group g_0 is allowed to be directly enclosed in an ambient of group g. (We would expect $\mathcal{O}(g_0, g)$ to imply $\mathcal{I}(g_0, g)$.) We write $\mathcal{I}(g_0, G)$ to mean that $\mathcal{I}(g_0, g)$ for every g in G.

Our assumption is that the top-level process consists of a set of ambients put in parallel, and implicitly located within a "global" ambient of group @ (this property is preserved under reduction).

[2]For this reason, it is argued in [BCC01] that opening should be disallowed altogether.

[3]The *safe ambients* proposed in [LS00] additionally include the co-capabilities "co-in" and "co-out" which also have been used, e.g., in [BC01].

EXAMPLE 1.1. Consider the "Trojan horse" from [BC01], which stripped of all co-capabilities, and with each ambient annotated with a unique group, looks like

$$a[\mathsf{open}\ b.\mathsf{in}\ c]^A \mid b[\mathsf{in}\ a.\mathsf{in}\ d]^B \mid c[P \mid d[Q]^D]^C$$

At run-time, b enters a where it is opened, leaving us with the configuration

$$a[\mathsf{in}\ c \mid \mathsf{in}\ d]^A \mid c[P \mid d[Q]^D]^C$$

Now a can enter c, and from there further enter d. This might not have been what the "owner" of c had in mind when "allowing" a to enter, as by looking only at the "code" of a there is no sign of a desire to enter d.

Our type system detects this attack, as in order for the process to be well-typed we have to make the type assignment

$$a : \mathsf{amb}^A_{@CD}$$

with the interpretation that a is of group A and may be immediately enclosed in[4] either @, C, or (the sign of danger) D. As we have seen, this is in fact an exact estimate. Similarly, assuming that P and Q do nothing of interest, we have

$$c : \mathsf{amb}^C_{@} \text{ and } d : \mathsf{amb}^D_C$$

The type assigned to b is more sophisticated:

$$b : \mathsf{amb}^B_{@A}[A :\ ^C\mathsf{enter}(D)]$$

with the interpretation that b may be immediately enclosed in either @ or A, and that after being opened inside A the process thereby unleashed has behavior $^C\mathsf{enter}(D)$, that is it will steer its surrounding ambient A from being inside C into being inside D. We shall now briefly argue why the above is a valid typing for b, assuming the dynamic security constraint is given by $\mathcal{O}(B, A)$ and assuming an appropriate definition of the static security constraints. First note that in order for an ambient to enter a it must be a "sibling" of a and therefore inside a group that also a may be immediately enclosed in, that is (cf. the type of a) inside either @, C, or D. Therefore the capability $\mathsf{in}\ a$ can be given behavior $^{@CD}\mathsf{enter}(A)$; and combined with similar reasoning for $\mathsf{in}\ d$ the process inside b can be assigned the behavior

$$^{@CD}\mathsf{enter}(A).\,^C\mathsf{enter}(D).$$

As b is initially within @, we read that b moves from @ to A and then suddenly shows up in C... the explanation being that b has been *opened* while inside A. We infer that b does in fact behave as predicted by its type: it can be enclosed in only @ or A, and after being opened[5] the process unleashed, to be executed by A, will behave as $^C\mathsf{enter}(D)$. □

EXAMPLE 1.2. Consider the variant of Example 1.1, also given in [BC01] (again we omit co-capabilities):

$$a[\mathsf{in}\ c]^A \mid b[\mathsf{in}\ a.\mathsf{out}\ a.\mathsf{in}\ d]^B \mid c[P \mid d[Q]^D]^C$$

Here one possible execution sequence is that b is carried by a into c, at which point b exits a and enters d—still, this might not be what the owner of c had in mind. The least typing assignment is given by

[4] We shall write $g_1 \ldots g_n$ for $\{g_1, \ldots, g_n\}$, and write "inside g" when we really mean "inside an ambient of group g".

[5] Which can only happen at one point, so we get exactly as precise information as if b had contained the process $\mathsf{in}\ a.\mathsf{co\text{-}open}\ b.\mathsf{in}\ d$.

$a : \mathsf{amb}^A_{@C}$, $b : \mathsf{amb}^B_{@ACD}$, $c : \mathsf{amb}^C_{@}$, $d : \mathsf{amb}^D_C$.

Thus the type of a does not reveal that it carries with it a process entering d; instead one has to examine the type of b which cannot be typed unless $\mathcal{I}(B, D)$ is allowed. This is in some sense similar to what [CGG00] would do, whereas the attack is immediately detected (but using co-capabilities) in [BC01]. □

EXAMPLE 1.3. As a final example, consider the firewall first presented in [CG98]:

$$w[k[\mathsf{out}\ w.\mathsf{in}\ k'.\mathsf{in}\ w]^B \mid \mathsf{open}\ k'.\mathsf{open}\ k''.P]^A \mid k'[\mathsf{open}\ k.k''[Q]^D]^C$$

This process is deterministic: k will exit w and enter k' where it is dissolved; then k' will enter w where it is dissolved and afterwards k'' (carried into w by k') is also dissolved. We thus need the security constraints to allow:

$$\mathcal{O}(B, C), \mathcal{O}(C, A), \mathcal{O}(D, A)$$

$$\mathcal{I}(B, A), \mathcal{I}(D, C), \mathcal{I}(B, C), \mathcal{I}(C, A), \mathcal{I}(D, A), \mathcal{I}(A, @), \mathcal{I}(B, @), \mathcal{I}(C, @).$$

Assuming that P and Q do nothing of interest, we can type the ambients as follows:

$$\begin{aligned} E(k) &= \mathsf{amb}^B_{AC@}[C : {}^{@}\mathsf{enter}(A)] \\ E(k') &= \mathsf{amb}^C_{A@}[A : \varepsilon] \\ E(k'') &= \mathsf{amb}^D_{AC}[A : \varepsilon] \\ E(w) &= \mathsf{amb}^A_{@} \end{aligned}$$

In the non-causal analysis of [NNHJ99], A as well as C can contain all of A, B, C, D. As several other analyses in the recent literature, we are thus much more precise. □

The rest of this report is organized as follows: Sect. 2 defines the ambient language and its semantics; Sect. 3 defines the entities used by the type system which is then presented in Sect. 4 and shown semantically sound in Sect. 5. Sect. 6 shows that type checking is decidable; Sect. 7 briefly sketches the more difficult problem of type inference.

Acknowledgments. This report documents research done mainly in 2001 while I was at Boston University, supported by NSF EIA grant 9806745/9806746/9806747/9806835 and partially supported by Sun grant EDUD-7826-990410-US; also some part was done in 2002 when at Heriot-Watt University, supported by EC grant IST-2001-33477. I would like to thank Michele Bugliesi, Assaf Kfoury, and Santiago Pericas-Geertsen for inspiring discussions, and the latter two for commenting upon a short version of this paper.

2 The Language

Figure 1 shows the syntax of our language **AC**. A process $P \in \mathsf{Proc}$ is basically as in [CG99]: there are constructs for parallel composition ($P_1 \mid P_2$), replication ($!P$), restriction ($(\nu n : \tau).P$); and there are also constructs for input (where the names $n_1 \ldots n_k$ must be distinct) and output[6]. Note that for the binding constructs, the name n being bound is annotated with a type (to be defined in Sect. 3).

[6]Note that communication is asynchronous, in that an outputting process has no "continuation"; a communication can thus (cf. the metaphor in [Car99]) be viewed as the placement, and subsequent removal, of a Post-It note on a message board. On the other hand, we believe that our development would carry through (with the obvious modifications) also for an extension of **AC** allowing synchronous communication.

Let $n, m \in \mathsf{Names}$ range over names.
Let $\xi, \chi \in \mathsf{Tags}$ range over tags.

Expressions

$$M \in \mathsf{Exp} \quad ::= \quad n \mid \mathsf{in}\ M \mid \mathsf{out}\ M \mid \mathsf{open}\ M \mid \epsilon \mid M_1.M_2 \mid \ldots$$

Processes

$$\begin{array}{rcl} P, Q, R \in \mathsf{Proc} & ::= & \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu n : \tau).P \mid M.P \mid M[\![P]\!]^{\xi} \\ & \mid & (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \qquad (k \geqslant 0) \end{array}$$

When there is no ambiguity we write M for $M.\mathbf{0}$.

Figure 1. Syntax of AC.

An expression $M \in \mathsf{Exp}$ denotes a computation over a domain that includes primarily ambient names and (paths of) capabilities, but (as spelt out in detail in [AKPG01]) it could also include simple values (like integers) or functions.

Since the formulation of semantic soundness (Sect. 5) requires us to distinguish between ambients with the same name (as is customary in approaches using flow logic, such as [NNHJ99]), we shall annotate each ambient $M[\![P]\!]^{\xi}$ with a *tag* ξ. This conforms with the notation used in the Introduction, assuming that there exists a function

$$\mathit{group} : \mathsf{Tags} \to \mathsf{Grps}$$

allowing one to read the group of an ambient from its tag. We say that a process P is *uniquely tagged* if whenever $M_1[\![P_1]\!]^{\xi}$ and $M_2[\![P_2]\!]^{\chi}$ occur at different positions in P then $\xi \neq \chi$.

Definition 2.1. Given P and P', we say that $P \equiv_{\mathrm{t}} P'$ (to be read "equal modulo group") if *(i)* P and P' are equal except for the tags; and *(ii)* if P at some position has tag ξ then P' at the same position has a tag χ with $\mathit{group}(\xi) = \mathit{group}(\chi)$. □

For instance, $n[\![\mathbf{0}]\!]^{\xi} \equiv_{\mathrm{t}} n[\![\mathbf{0}]\!]^{\chi}$ if $\mathit{group}(\xi) = \mathit{group}(\chi)$.

The set of names occurring free in P is denoted $\mathsf{fn}(P)$; the set of all names occurring in P is denoted $\mathsf{names}(P)$.

Definition 2.2. We say that a process P is non-conflicting with a set of names X if *(i)* no name is bound more than once in P, and *(ii)* a name bound in P does not occur in X. □

For all P and X, we can clearly find P' such that P' is non-conflicting with X and such that P' and P are equal modulo consistent renaming of bound names.

2.1 Operational Semantics

The semantics of **AC** is presented in Fig. 2. Before an expression M can be communicated to another process it must evaluate to a value V, using an evaluation relation $M \longrightarrow M'$ which is left unspecified.

We write $P_1 \equiv P_2$ to denote that P_1 and P_2 are equivalent, modulo consistent renaming of bound names (which may be needed to apply (Red Comm), as this rule has a side condition preventing name capture) and modulo "syntactic rearrangement". The relation is given as the least

one satisfying the clauses presented in Fig. 3 and is as in [CG99], except that (in order to establish Lemma 5.5) we omit the rule $!P \equiv P \mid !P$ and instead allow this "unfolding" to take place via the rule (Red Repl) (as also seen in, e.g., [NNS00]) To enable the reduct to be uniquely tagged, this rule actually permits $!P$ to be unfolded into $P' \mid !P$ where P' is a "copy" of P (cf. Definition 2.1).

We write $P_1 \xrightarrow{\ell} P_2$ if P_1 reduces in one step to P_2 by performing "an action described by ℓ". Note that the definition of this relation deviates from the standard in that in (Red Open) and (Red Comm) we provide the surrounding ambient so as to record in ℓ *where* the opening or communication has taken place.

We use a notion of "process evaluation contexts" to succinctly describe the place in a process where an expression (Red MctxtP) or subprocess (Red PctxtP) is reduced. Note that $P \xrightarrow{\ell} Q$ does not imply that $M.P \xrightarrow{\ell} M.Q$ since M must evaluate to a capability which then is executed before P can be activated.

A couple of elementary properties of the semantics are stated below:

Lemma 2.3. *If $P \equiv Q$ then P and Q contain the same tags, and P is uniquely tagged iff Q is.* □

The proof is a simple induction in the derivation of $P \equiv Q$.

Lemma 2.4. *If $P \xrightarrow{\ell} Q$ then all tags in $dom(\ell)$ occur in P.* □

The proof is a simple induction in the derivation of $P \xrightarrow{\ell} Q$, making use of Lemma 2.3.

3 Types and Behaviors

Our type system assigns types ($\tau \in$ Typ) to expressions and behaviors ($b \in$ Beh) to processes; these entities are recursively defined in Fig. 4 using auxiliary notions such as *actions* and *behavior contexts*.

In the definition of actions, we use H to range over *upwards closed* sets of groups, where G is upwards closed if $g \in G$ and $\mathcal{O}(g, g')$ implies $g' \in G$. The intuition is that if an ambient n can be directly enclosed in g, and g' can open g, then n might also be directly enclosed in g'. We let $g^{\uparrow}$ denote the least upwards closed set containing g.

EXAMPLE 3.1. In Example 1.1, we have $B^{\uparrow} = \{A, B\}$.

In Example 1.3, we have $A^{\uparrow} = \{A\}$ and $B^{\uparrow} = \{A, B, C\}$ and $C^{\uparrow} = \{A, C\}$ and $D^{\uparrow} = \{A, D\}$. □

Rather than defining behaviors syntactically, using certain constructors (as in [AKPG01]), we shall take a more semantic approach and define a behavior as a regular (possibly infinite) and non-empty set of (finite) traces. (The latter condition is needed in the proof of semantic soundness, which requires behaviors to be inhabited.) This gives the user more freedom in specification, and fits well with type checking which (Sect. 6) is carried out using finite automata. Nevertheless, it is still convenient to define certain semantic operators[7] on behaviors:

$$\begin{aligned} \varepsilon &= \{\bullet\} \\ a.b &= \{a \diamond tr \mid tr \in b\} \\ b_1 \mid b_2 &= \bigcup_{tr_1 \in b_1, tr_2 \in b_2} tr_1 \parallel tr_2 \end{aligned}$$

[7] Corollary 6.15 will show that these operators do indeed map regular sets into regular sets.

Values

$$V \quad ::= \quad n \mid \mathsf{in}\ n \mid \mathsf{out}\ n \mid \mathsf{open}\ n \mid \epsilon \mid V_1.V_2 \mid \ldots$$

Reduction Labels

ℓ	$::=$	ϵ	only "internal computation" is performed
	$\mid$	$\xi : \mathsf{enter}\ \chi$	the ambient ξ is steered into the ambient χ
	$\mid$	$\xi : \mathsf{exit}\ \chi$	the ambient ξ is steered out of the ambient χ
	$\mid$	$\xi : \mathsf{open}\ \chi$	a process controlling the ambient ξ opens the ambient χ
	$\mid$	$\xi : \mathsf{comm}\ \sigma$	inside the ambient ξ values of type σ are communicated

Process Evaluation Contexts

$$\mathcal{PC} \quad ::= \quad \Box_p \mid (\mathcal{PC} \mid P) \mid (\nu n : \tau).\mathcal{PC} \mid n[\![\mathcal{PC}]\!]^{\xi}$$
$$\mid \quad \Box_e.P \mid \Box_e[\![P]\!]^{\xi} \mid \langle V_1, \ldots, V_{i-1}, \Box_e, M_{i+1}, \ldots, M_k \rangle$$

Notation: if the hole in $\mathcal{PC}$ is an expression hole $\Box_e$ then $\mathcal{PC}\lfloor M \rfloor_e$ is the process resulting from replacing this hole with M; and if the hole in $\mathcal{PC}$ is a process hole $\Box_p$ then $\mathcal{PC}\lfloor P \rfloor_p$ is the process resulting from replacing this hole with P.

Reduction Rules

In (Red Comm) we demand that $(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P$ is non-conflicting with $\mathsf{names}(V_1, \ldots, V_k)$.

$$m[\![\mathsf{in}\ n.P \mid Q]\!]^{\xi} \mid n[\![R]\!]^{\chi} \xrightarrow{\xi:\mathsf{enter}\ \chi} n[\![m[\![P \mid Q]\!]^{\xi} \mid R]\!]^{\chi} \qquad (\text{Red In})$$

$$n[\![m[\![\mathsf{out}\ n.P \mid Q]\!]^{\xi} \mid R]\!]^{\chi} \xrightarrow{\xi:\mathsf{exit}\ \chi} m[\![P \mid Q]\!]^{\xi} \mid n[\![R]\!]^{\chi} \qquad (\text{Red Out})$$

$$m[\![\mathsf{open}\ n.P \mid n[\![Q]\!]^{\chi} \mid R]\!]^{\xi} \xrightarrow{\xi:\mathsf{open}\ \chi} m[\![P \mid Q \mid R]\!]^{\xi} \qquad (\text{Red Open})$$

$$m[\![(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle V_1 \ldots V_k \rangle \mid Q]\!]^{\xi} \xrightarrow{\xi:\mathsf{comm}\ \times(\tau_1, \ldots, \tau_k)} m[\![P[n_i := V_i] \mid Q]\!]^{\xi} \qquad (\text{Red Comm})$$

$$!P \xrightarrow{\epsilon} P' \mid !P \quad \text{if } P' \equiv_{\mathrm{t}} P \qquad (\text{Red Repl})$$

$$\text{If } M_1 \longrightarrow M_2 \text{ then } \mathcal{PC}\lfloor M_1 \rfloor_e \xrightarrow{\epsilon} \mathcal{PC}\lfloor M_2 \rfloor_e \qquad (\text{Red MctxtP})$$

$$\text{If } P \xrightarrow{\ell} Q \text{ then } \mathcal{PC}\lfloor P \rfloor_p \xrightarrow{\ell} \mathcal{PC}\lfloor Q \rfloor_p \qquad (\text{Red PctxtP})$$

$$\text{If } P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \text{ then } P' \xrightarrow{\ell} Q' \qquad (\text{Red } \equiv)$$

Figure 2. Operational Semantics.

Here $\bullet$ denotes the empty sequence, $a \diamond tr$ denotes the result of putting a in front of tr, $tr_1 \diamond tr_2$ denotes concatenation, and $tr_1 \parallel tr_2$ denotes all traces that can be formed by arbitrarily interleaving[8] tr_1 with tr_2. The operator $\mid$ is associative and commutative, with $\{\bullet\}$ as neutral element. When there is no ambiguity, we write a for $a.\varepsilon$; and if $tr = a_1 \ldots a_k$ we write $tr.b$ for $a_1 \ldots a_k.b$.

An ambient n has a type of the form $\mathsf{amb}^{g}_{H}[\{g_i : b_i\}_{i \in I}]$, which (cf. the Introduction) should be

[8] To be more precise: *(i)* $\bullet \in \bullet \parallel \bullet$; *(ii)* $a \diamond tr' \in tr_1 \parallel tr_2$ iff $tr_1 = a \diamond tr_1'$ with $tr' \in tr_1' \parallel tr_2$ or $tr_2 = a \diamond tr_2'$ with $tr' \in tr_1 \parallel tr_2'$.

$P \equiv P$	(Struct Refl)
If $P \equiv Q$ then $Q \equiv P$	(Struct Symm)
If $P \equiv Q$ and $Q \equiv R$ then $P \equiv R$	(Struct Trans)
If $P \equiv Q$ then $(\nu n : \tau).P \equiv (\nu n : \tau).Q$	(Struct Res)
If $P \equiv Q$ then $P \mid R \equiv Q \mid R$	(Struct Par)
If $P \equiv Q$ then $!P \equiv !Q$	(Struct Repl)
If $P \equiv Q$ then $M[\![P]\!]^{\xi} \equiv M[\![Q]\!]^{\xi}$	(Struct Amb)
If $P \equiv Q$ then $M.P \equiv M.Q$	(Struct Action)
If $P \equiv Q$ then $(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \equiv (n_1 \ldots n_k : \tau_1 \ldots \tau_k).Q$	(Struct Input)
$P \mid Q \equiv Q \mid P$	(Struct ParComm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct ParAssoc)
$(\nu n_1 : \tau_1).(\nu n_2 : \tau_2).P \equiv (\nu n_2 : \tau_2).(\nu n_1 : \tau_1).P$ if $n_1 \neq n_2$	(Struct ResRes)
$(\nu n : \tau).(P \mid Q) \equiv P \mid (\nu n : \tau).Q$ if $n \notin \mathsf{fn}(P)$	(Struct ResPar)
$(\nu n : \tau).m[\![P]\!]^{\xi} \equiv m[\![(\nu n : \tau).P]\!]^{\xi}$ if $n \neq m$	(Struct ResAmb)
$P \mid \mathbf{0} \equiv P$	(Struct ZeroPar)
$(\nu n : \tau).\mathbf{0} \equiv \mathbf{0}$	(Struct ZeroRes)
$!\mathbf{0} \equiv \mathbf{0}$	(Struct ZeroRepl)
$\epsilon.P \equiv P$	(Struct ε)
$(M.M').P \equiv M.(M'.P)$	(Struct .)
$P \equiv Q$ if P and Q are equal modulo consistent renaming of bound names	(Struct α-rename)

Figure 3. Structural Congruence.

read as follows: it has group g, can be directly enclosed inside ambients of groups belonging to H, and after being opened inside g_i it behaves as b_i.

A capability has a type of the form $\mathsf{cap}[B]$ where B is a behavior context, that is a "behavior with a hole inside" (a convenient device which was introduced in [AKPG01] and used also in [GYY01]). The idea can best be illustrated by an example: if n has type $\mathsf{amb}^{g}_{H}[g_0 : b_0]$ and P has behavior b, then the process $\mathsf{open}\ n.P$ will open g and then run b_0 in parallel with b. And in fact, referring to Fig. 7, ${}_{g_0}\mathsf{open}(g).(b_0 \mid b)$ is the result of (Proc Action) plugging b into $\mathsf{cap}[{}_{g_0}\mathsf{open}(g).(b_0 \mid \Box)]$ which by (Exp Open) is the type of $\mathsf{open}\ n$. To be more precise, given B and b we define the behavior $B\lfloor b \rfloor$ by (recursively) stipulating

$$\begin{aligned} \Box\lfloor b \rfloor &= b \\ (a.B)\lfloor b \rfloor &= a.B\lfloor b \rfloor \\ (b_0 \mid B)\lfloor b \rfloor &= b_0 \mid B\lfloor b \rfloor \end{aligned}$$

(Note that the operators on the left hand sides are syntactic entities, whereas the ones on the right hand sides are the semantic operators defined above.) Similarly, given B and B_1 we define the behavior context $B\lfloor B_0 \rfloor$ by (recursively) stipulating

$$\begin{aligned} \Box\lfloor B_0 \rfloor &= B_0 \\ (a.B)\lfloor B_0 \rfloor &= a.B\lfloor B_0 \rfloor \\ (b_0 \mid B)\lfloor B_0 \rfloor &= b_0 \mid B\lfloor B_0 \rfloor \end{aligned}$$

(Note that both on the left hand side and the right hand side, the operators are syntactic entities.) In Appendix A, we prove

Lemma 3.2. *Given behavior contexts B_1, B_2, and behavior b. Then $B_1\lfloor B_2\lfloor b \rfloor \rfloor = (B_1\lfloor B_2 \rfloor)\lfloor b \rfloor$.* □

Actions

$$
\begin{array}{rll}
a \in \mathsf{Act} ::= & {}^{H}\mathsf{enter}(g) & \text{steers an ambient from } H \text{ to } g \\
\mid & {}^{g}\mathsf{exit}(H) & \text{steers an ambient from } g \text{ to } H \\
\mid & {}_{G}\mathsf{open}(g) & \text{if executed in } G\text{, opens } g \\
\mid & \mathsf{put}(\sigma) & \text{output tuple of type } \sigma \\
\mid & \mathsf{get}(\sigma) & \text{input tuple of type } \sigma
\end{array}
$$

Traces

$$tr \in \mathsf{Trace} ::= a^* \qquad \text{finite sequence of actions}$$

Behaviors

$$b \in \mathsf{Beh} \subseteq \mathcal{P}(\mathsf{Trace}) \qquad \text{non-empty regular set of traces}$$

Behavior Contexts

$$B \in \mathsf{BehCont} ::= \Box \mid a.B \mid (b \mid B)$$

Behavior Rows

$$br \in \mathsf{BehRows} ::= \{g_i : b_i\}_{i \in I} \qquad \text{behaves as } b_i \text{ when opened in } g_i$$

We shall write $\emptyset$ if I is empty, and $\{g_i : b_i\}$ if I is a singleton $\{i\}$.

Tuples

$$\sigma \in \mathsf{Tuple} ::= \times(\tau_1, \ldots, \tau_k) \qquad (k \geqslant 0)$$

When there is no ambiguity, we write τ_1 if $k = 1$.

Types

$$
\begin{array}{rll}
\tau \in \mathsf{Typ} ::= & \mathsf{amb}^{g}_{H}[br] & \text{type of ambient name} \\
\mid & \mathsf{cap}[B] & \text{type of capability} \\
\mid & \mathsf{int} \mid \mathsf{real} & \\
\mid & \ldots & \text{other types (optional)}
\end{array}
$$

In $\mathsf{amb}^{g}_{H}[\{g_i : b_i\}_{i \in I}]$, we demand that $H \neq \emptyset$ and $\mathcal{I}(g, H)$ and $\forall i \in I$: $\mathcal{O}(g, g_i)$.
If $I = \emptyset$ we may write amb^{g}_{H}; if I is a singleton $\{i\}$ we may write $\mathsf{amb}^{g}_{H}[g_i : b_i]$.

Figure 4. Syntax of Types and Behaviors.

We shall employ the notion of *level*: an entity has level i if i is an upper bound of the depth of nested occurrences of $\mathsf{amb}\_[\_]$ or $\mathsf{cap}[\_]$ within it. (We use "_" to stand for an arbitrary entity of the appropriate kind.) Example: $\mathsf{put}(\mathsf{cap}[\{\mathsf{put}(\mathsf{amb}^{g}_{H})\} \mid \Box])$ has level two. Note that an entity that has level i also has level j for all $j > i$.

If $H_1 \subseteq H_2$ then ${}^{H_1}\mathsf{enter}(g) \leqslant {}^{H_2}\mathsf{enter}(g)$	(Act Enter)
If $H_1 \subseteq H_2$ then ${}^{g}\mathsf{exit}(H_1) \leqslant {}^{g}\mathsf{exit}(H_2)$	(Act Exit)
If $G_2 \subseteq G_1$ then ${}_{G_1}\mathsf{open}(g) \leqslant_{G_2} \mathsf{open}(g)$	(Act Open)
If $\sigma_1 \leqslant \sigma_2$ then $\mathsf{put}(\sigma_1) \leqslant \mathsf{put}(\sigma_2)$	(Act Put)
If $\sigma_2 \leqslant \sigma_1$ then $\mathsf{get}(\sigma_1) \leqslant \mathsf{get}(\sigma_2)$	(Act Get)

Figure 5. The Ordering on Actions

3.1 Ordering Relations

In the subsequent paragraphs we define relations $b_1 \leqslant b_2$ and $\tau_1 \leqslant \tau_2$. The definitions are mutually recursive, yet well-defined: a relation on level i types induces a relation on level i tuples which induces a relation on level i actions which induces a relation on level i traces which induces a relation on level i behaviors which induces a relation on level i behavior rows and contexts which in turn induces a relation on level $i+1$ types.

The relation $\leqslant$ on tuples is defined pointwise from the relation $\leqslant$ on types: $\times(\tau_1, \ldots, \tau_k) \leqslant \times(\tau'_1, \ldots, \tau'_{k'})$ iff $k = k'$ and for all $i \in \{1 \ldots k\}$: $\tau_i \leqslant \tau'_i$ (thus tuples with different arity are incompatible).

The relation $a_1 \leqslant a_2$, with the intuitive interpretation that a_2 is more "permissive" than a_1, is defined as the least one satisfying the clauses presented in Fig. 5 which can be summarized by stating that the constructors have the following polarity:

$$^{\oplus}\mathsf{enter}(=) \quad {}^{=}\mathsf{exit}(\oplus) \quad {}_{\ominus}\mathsf{open}(=) \quad \mathsf{put}(\oplus) \quad \mathsf{get}(\ominus)$$

Concerning the polarity of ${}^{g}\mathsf{exit}(H)$, the intuition is that if the ambient as a result of leaving g will enter an ambient whose group is in H, this group also belongs to any set containing H; similarly for ${}^{H}\mathsf{enter}(g)$. For ${}_{G}\mathsf{open}(g)$, the intuition is that if a process will open g provided it is in an ambient with group in G, it will also open g whenever it is in an ambient with a group belonging to a subset of G. Concerning the actions for communication, we have that $\mathsf{put}(\mathsf{int}) \leqslant \mathsf{put}(\mathsf{real})$, since a process that sends an integer thereby also sends a real number, and $\mathsf{get}(\mathsf{real}) \leqslant \mathsf{get}(\mathsf{int})$, since a process that accepts a real number also will accept an integer. Thus output is covariant and input is contravariant, while in other systems found in the literature it is the other way round—the reason for this discrepancy is that we take a *descriptive* rather than a *prescriptive* point of view. From a prescriptive point of view, a channel that allows the writing of real numbers also allows the writing of integers, and a channel that allows the reading of integers also allows the reading of real numbers.

The relation $b_1 \leqslant b_2$, with the intuitive interpretation that b_2 is more "permissive" than b_1, is defined by stipulating that $b_1 \leqslant b_2$ iff for all $tr_1 \in b_1$ there exists $tr_2 \in b_2$ such that $tr_1 \leqslant tr_2$. Here the relation $tr_1 \leqslant tr_2$ is the pointwise extension of the relation $a_1 \leqslant a_2$ (note that if $tr_1 \leqslant tr_2$ then tr_1 and tr_2 have the same length).

The relation $\leqslant$ on behavior rows is defined as follows:

Definition 3.3. With $br = \{g_i : b_i\}_{i \in I}$ and $br' = \{g'_j : b'_j\}_{j \in J}$, $br \leqslant br'$ holds iff for all $j \in J$ there exists $i \in I$ such that $g_i = g'_j$ and $b_i \leqslant b'_j$. □

We write $\equiv$ for the equivalence relation induced by $\leqslant$. That is, we write $b_1 \equiv b_2$ whenever $b_1 \leqslant b_2$

$\tau \leqslant \tau$	(Typ Refl)
If $\tau_1 \leqslant \tau_2$ and $\tau_2 \leqslant \tau_3$ then $\tau_1 \leqslant \tau_3$	(Typ Trans)
$\mathsf{int} \leqslant \mathsf{real}$	(Typ Base)
If $br_1 \leqslant br_2$ and $br_2 \leqslant br_1$ then $\mathsf{amb}^g_H[br_1] \leqslant \mathsf{amb}^g_H[br_2]$	(Typ Amb)
If $B_1 \leqslant B_2$ then $\mathsf{cap}[B_1] \leqslant \mathsf{cap}[B_2]$	(Typ Cap)

Figure 6. The Subtyping Relation

and $b_2 \leqslant b_1$. Note that $b_1 \equiv b_2$ does not necessarily imply $b_1 = b_2$ (for instance, let $b_1 = \{a_1\}$ and let $b_2 = \{a_1, a_2\}$ with $a_2 \leqslant a_1$).

The relation $\leqslant$ on behavior contexts is defined as follows:

Definition 3.4. $B_1 \leqslant B_2$ holds iff for all level 0 behaviors b we have $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$. □

The restriction to level 0 behaviors is formally needed, in order for the relation on level i behavior contexts to be determined by the relation on level i behaviors, but is not essential: Lemma 3.12 below will tell us that if $B_1 \leqslant B_2$ then $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$ holds for all b (moreover, for all B_1, B_2 there exists b_0 such that testing $B_1 \leqslant B_2$ amounts to testing $B_1\lfloor b_0 \rfloor \leqslant B_2\lfloor b_0 \rfloor$).

The relation $\tau_1 \leqslant \tau_2$, with the intuitive interpretation that an expression of type τ_1 also has type τ_2, is defined as the least one satisfying the clauses presented in Fig. 6, which can be summarized by stating that $\mathsf{int} \leqslant \mathsf{real}$ and that we have the polarity $\mathsf{cap}[\oplus]$. Concerning the polarity of the type $\mathsf{amb}^g_H[br]$, it turns out that the proof of subject reduction reveals that this type must be covariant as well as contravariant in H and br. We could thus (as in [AKPG01]) play the trick of [Zim00] (akin to the "split types" of [BPG00] for an object-oriented calculus) and split each of these arguments into two, one contravariant and the other covariant. But to keep things simple, we refrain from doing so.

The following basic results are straightforward to verify:

Lemma 3.5. *The relations $\leqslant$ defined above are preorders, that is reflexive and transitive.* □

Lemma 3.6. *The operators "$\mid$" and "." on behaviors respect the relation $\leqslant$; thus $\equiv$ is a congruence on behaviors wrt. these operators. Moreover, modulo $\equiv$ it holds that "$\mid$" is associative and commutative, with ε as neutral element.* □

Lemma 3.7. *If $b_1 \leqslant b_2$ then for all behaviour contexts B it holds that $B\lfloor b_1 \rfloor \leqslant B\lfloor b_2 \rfloor$.* □

Lemma 3.8. *Given action a and behaviors b_1 and b_2. Then $a.(b_1 \mid b_2) \leqslant (a.b_1) \mid b_2$.* □

Judging from Definition 3.4, the relation $B_1 \leqslant B_2$ seems impossible to decide, since it apparently involves plugging in an infinite number of behaviors. Fortunately for type checking (as to be done in Sect. 6), Lemma 3.12 (stated below) tells us that it suffices to plug in a single (level 0) behavior $\{\mathsf{test}\}$, provided test *tests* B_1 and B_2 according to the following definition:

Definition 3.9. Given a behavior context B. We say that an action a tests B if a is incomparable with all actions occurring as part of B. To be precise:

- a tests $\Box$
- a tests $a'.B$ provided a tests B and neither $a \leqslant a'$ nor $a' \leqslant a$ does hold

- a tests $b \mid B$ provided a tests B and for all $tr \in b$ and all a' occurring in tr neither $a \leqslant a'$ nor $a' \leqslant a$ does hold. □

Lemma 3.10. *Given a finite set of behavior contexts, we can always construct an action a that tests these behavior contexts.*

Similarly, given a finite set of behaviors we can always construct an action a that does not occur in any trace contained in these behaviors. □

Proof. First note that any behavior b, being a regular set of traces, can contain only a finite number of different actions. Similarly for a behavior context B.

Then choose, for instance, $a = {}^{\emptyset}\mathsf{enter}(g^*)$ where g^* is a group not occurring in any of these actions. □

We have the following auxiliary result, to be proved in Appendix A:

Lemma 3.11. *Given behavior context B, and an action* test *such that* test *tests B. For all behaviors b and traces tr, $tr \in B\lfloor b \rfloor$ holds if and only if there exists tr_1, tr_2, tr_0 such that $tr_1 \diamond \mathsf{test} \diamond tr_2 \in B\lfloor \{\mathsf{test}\} \rfloor$ and $tr_0 \in \{tr_2\} \mid b$ and $tr = tr_1 \diamond tr_0$.* □

We are now ready to state

Lemma 3.12. *Given B_1 and B_2, we can construct an action* test *of level zero such that the following conditions are equivalent:*

(a) $B_1 \leqslant B_2$

(b) $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$ *for all b (regardless of level)*

(c) $B_1\lfloor \{\mathsf{test}\} \rfloor \leqslant B_2\lfloor \{\mathsf{test}\} \rfloor$. □

Proof. First construct, by Lemma 3.10, an action test such that test tests B_1 and B_2. We now show that (c) implies (b), the only non-trivial part of the lemma.

For this purpose, assume that b has been given. Let tr belong to $B_1\lfloor b \rfloor$, then our task is to find $tr^+ \in B_2\lfloor b \rfloor$ with $tr^+ \geqslant tr$. By Lemma 3.11, there exists tr_1, tr_2, tr_0 such that $tr_1 \diamond \mathsf{test} \diamond tr_2 \in B_1\lfloor \mathsf{test} \rfloor$ and $tr_0 \in \{tr_2\} \mid b$ and $tr = tr_1 \diamond tr_0$. Using our assumption (c), there exists a trace $tr' \in B_2\lfloor \mathsf{test} \rfloor$ with $tr_1 \diamond \mathsf{test} \diamond tr_2 \leqslant tr'$. We can write $tr' = tr_1^+ \diamond \mathsf{test} \diamond tr_2^+$ with $tr_1 \leqslant tr_1^+$ and $tr_2 \leqslant tr_2^+$, and clearly there exists $tr_0^+ \in \{tr_2^+\} \mid b$ with $tr_0 \leqslant tr_0^+$. Define $tr^+ = tr_1^+ \diamond tr_0^+$. By Lemma 3.11 we infer that $tr^+ \in B_2\lfloor b \rfloor$. As clearly $tr \leqslant tr^+$, this is as desired. □

3.2 Predicates on Traces

Given an ambient n, controlled by a process P and residing within an ambient whose group belongs to H, we want to estimate the destination of n after (partial) execution of P. For this purpose, we define the upwards closed sets $Dest(H, a)$ and $Dest(H, tr)$ as follows:

$$\begin{aligned}
Dest(H, {}^{H_0}\mathsf{enter}(g)) &= \text{if } H \cap H_0 \neq \emptyset \text{ then } g^{\uparrow} \text{ else } \emptyset \\
Dest(H, {}^{g}\mathsf{exit}(H_0)) &= \text{if } g \in H \text{ then } H_0 \text{ else } \emptyset \\
Dest(H, a) &= H \text{ otherwise} \\
Dest(H, \bullet) &= H \\
Dest(H, a \diamond tr) &= Dest(Dest(H, a), tr)
\end{aligned}$$

The idea behind the first line is that an ambient residing inside H can only enter an ambient g if they are "siblings", a necessary condition for which is that $H \cap H_0 \neq \emptyset$ where H_0 are the possible surroundings for g.

Lemma 3.13. *$Dest(H, tr)$ is monotone in H as well as in tr. Moreover, $Dest(\emptyset, tr) = \emptyset$ for all tr.* □

Note that the monotonicity in tr hinges on the polarities ${}^{\oplus}\mathsf{enter}(=)$ and ${}^{=}\mathsf{exit}(\oplus)$.

Lemma 3.14. $Dest(H, tr_1 \diamond tr_2) = Dest(Dest(H, tr_1), tr_2)$. □

Proof. Induction in the length of tr_1. If $tr_1 = \bullet$, the claim is trivial. If $tr_1 = a \diamond tr_1'$, the result follows from the following calculation (where the second equality is due to the induction hypothesis): $Dest(H, (a \diamond tr_1') \diamond tr_2) = Dest(Dest(H, a), tr_1' \diamond tr_2) = Dest(Dest(Dest(H, a), tr_1'), tr_2) = Dest(Dest(H, tr_1), tr_2)$. □

Lemma 3.15. *Assume that $g \in Dest(H, a)$. Then there exists $g_0 \in H$ such that $g \in Dest({g_0}^{\uparrow}, a)$.* □

Proof. By inspection of the various cases. □

We shall have particular interest in *feasible* traces, the intuition being that such a trace can execute "on its own"[9].

Definition 3.16. We say that a trace tr is *saturated* if all occurrences of $\mathsf{put}(\_)$ and $\mathsf{get}(\_)$ in tr come in pairs, with $\mathsf{put}(\_)$ immediately preceding $\mathsf{get}(\_)$. □

Definition 3.17. We say that a trace tr is *feasible* from H if

1. $Dest(H, tr) \neq \emptyset$, and
2. tr is saturated. □

Note that if $H \neq \emptyset$ then $\bullet$ is feasible from H.

EXAMPLE 3.18. The trace ${}^{@_{CD}}\mathsf{enter}(A)\ {}^{C}\mathsf{enter}(D)$ is not feasible from $\{@\}$ (cf. Example 1.1). For $Dest(@, tr) = Dest(Dest(@, {}^{@_{CD}}\mathsf{enter}(A)), {}^{C}\mathsf{enter}(D)) = Dest(A, {}^{C}\mathsf{enter}(D)) = \emptyset$. □

We now define a predicate, crucial for our type system, telling whether an ambient initially residing within H_0, and controlled by a process with behavior b, can be assigned the type $\mathsf{amb}^{g}_{H}[\{g_i : b_i\}_{i \in I}]$. For this to be the case, H must be an upper approximation of where the ambient can travel "on its own", and if opened when inside g_i then b_i must approximate the unleashed behavior; moreover, no type errors must arise during communication, and the "condition" of an opening action must be satified. Formally, we have

Definition 3.19 (The trace correctness predicate)**.** The relation

$$(H_0, b) \overset{g}{\rightsquigarrow} (H, \{g_i : b_i\}_{i \in I})$$

holds iff for all $tr_1 \diamond tr_2 \in b$ such that tr_1 is feasible from H_0 the following properties hold with $H_1 = Dest(H_0, tr_1)$:

1. $H_1 \subseteq H$, and
2. for all $i \in I$: $g_i \in H_1$ implies $\{tr_2\} \leqslant b_i$, and

[9] A non-feasible trace is still useful, since interleaving it with another trace may produce feasible traces.

3. if tr_2 takes the form $\mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2) \diamond tr_3$ then $\sigma_1 \leqslant \sigma_2$, and
4. if tr_2 takes the form ${}_G\mathsf{open}(\_) \diamond tr_3$ then $g \in G$. □

EXAMPLE 3.20. Looking back at Example 1.1, it is an instructive exercise to verify that

$$(@,\ {}^{CD}@_{\mathsf{enter}}(A).\,{}^{C}\mathsf{enter}(D)) \overset{B}{\rightsquigarrow} (A@, \{A :\ {}^{C}\mathsf{enter}(D)\})$$
$$(@,\ {}_{A}\mathsf{open}(B).({}^{C}\mathsf{enter}(D) \mid @_{\mathsf{enter}}(C))) \overset{A}{\rightsquigarrow} (CD@, \emptyset)$$

For the first line, we (cf. Example 3.18) only need to consider the cases $tr_1 = \bullet$ and $tr_1 = {}^{CD}@_{\mathsf{enter}}(A)$. □

EXAMPLE 3.21. Looking back at Example 1.2, it is easy to verify that it holds that

$$(@,\ @_{\mathsf{enter}}(C)) \overset{A}{\rightsquigarrow} (C@, \emptyset)$$
$$(@,\ {}^{C}@_{\mathsf{enter}}(A).\,{}^{A}\mathsf{exit}(C@).\,{}^{C}\mathsf{enter}(D)) \overset{B}{\rightsquigarrow} (ACD@, \emptyset)$$

□

EXAMPLE 3.22. Looking back at Example 1.3, it is instructive to verify that

$$(A,\ {}^{A}\mathsf{exit}(@).\,{}^{A}@_{\mathsf{enter}}(C).\,@_{\mathsf{enter}}(A)) \overset{B}{\rightsquigarrow} (AC@, \{C :\ @_{\mathsf{enter}}(A)\})$$
$$(@,\ {}_{C}\mathsf{open}(B).\,@_{\mathsf{enter}}(A)) \overset{C}{\rightsquigarrow} (A@, \{A : \varepsilon\})$$
$$(AC, \varepsilon) \overset{D}{\rightsquigarrow} (AC, \{A : \varepsilon\})$$
$$(@,\ {}_{A}\mathsf{open}(C).{}_{A}\mathsf{open}(D)) \overset{A}{\rightsquigarrow} (@, \emptyset)$$

For the first line, we (referring back to the terminology of Def. 3.19) only need to consider the cases $tr_1 = \bullet$, $tr_1 = {}^{A}\mathsf{exit}(@)$, and $tr_1 = {}^{A}\mathsf{exit}(@)\ {}^{A}@_{\mathsf{enter}}(C)$. This reflects that the action $@_{\mathsf{enter}}(A)$ cannot be performed by k before it gets opened. □

Some useful results about the trace correctness predicate, all to be proved in Appendix A:

Lemma 3.23. *If $(H_0, b) \overset{g}{\rightsquigarrow} (H, \{g_i : b_i\}_{i\in I})$ then $H_0 \subseteq H$, and for all $i \in I$ with $g_i \in H_0$ we have $b \leqslant b_i$.* □

Lemma 3.24. *The predicate $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ is anti-monotonic in H_0 and b, and monotonic in H and br.* □

Lemma 3.25. *Let tr be of the form either $\_\mathsf{enter}(\_)$, $\_\mathsf{exit}(\_)$, $\_\mathsf{open}(\_)$, or $\mathsf{put}(\_)\,\mathsf{get}(\_)$. Assume that*

$$(H_0, tr.b) \overset{g}{\rightsquigarrow} (H, br)$$

Then with $H_1 = Dest(H_0, tr)$ we also have

$$(H_1, b) \overset{g}{\rightsquigarrow} (H, br).$$

□

4 The Type System

Figure 7 defines judgments $E \vdash M : \tau$ and $\Delta, E \vdash_g P : b$, where E is an environment mapping names into types (we write $E, n : \tau$ for the environment E' that behaves as E except that it maps n into τ), where Δ is a *behavior pool* mapping tags into behaviors, and where g is the group of the ambient in which P is initially situated. Note that if $E \vdash M : \tau$ then there exists $\tau^- \leqslant \tau$ such that $E \vdash M : \tau^-$ is derived by a structural inference rule, and if $E \vdash P : b$ then there exists $b^- \leqslant b$ such that $E \vdash P : b^-$ is derived by a structural inference rule.

Already on p. 8 we saw how the rule (Exp Open) is designed to interact with the rule (Proc Action), similarly for the rules (Exp In) and (Exp Out). The side condition for the rule (Proc Repl) ensures that the behavior of a replicated process is in fact invariant under replication. The crux of the type system is the rule (Proc Amb), where the situation is that inside g we have an ambient M with tag ξ containing a process P. We must then be able to assign P the behavior $b = \Delta(\xi)$, when analyzed inside the group $g_0 = group(\xi)$. Moreover, it must hold that $(g^\uparrow, b) \overset{g_0}{\rightsquigarrow} (H, br)$ where $\mathsf{amb}^{g_0}_H[br]$ is the type of M, cf. the motivation for the trace correctness predicate given in Sect. 3.2.

EXAMPLE 4.1. Looking back at Example 1.1, the types provided for the ambients will indeed give rise to a valid type derivation. This is easy to verify, using the relations established in Example 3.20.

Similarly for Example 1.2 and Example 1.3, using the relations established in Example 3.21 and 3.22. □

For the static security constraint $\mathcal{I}(\_, \_)$, we have the following lemma:

Lemma 4.2. *Assume that inside P an ambient with group g_0 is directly enclosed in an ambient with group g. If P is typable then $\mathcal{I}(g_0, g)$ does hold.* □

Proof. From Lemma 3.23 we see that for the rule (Proc Amb) it holds that $g^\uparrow \subseteq H$, in particular that $g \in H$. From the requirement to an $\mathsf{amb}\_[\_]$ type we have $\mathcal{I}(g_0, H)$, in particular $\mathcal{I}(g_0, g)$. □

For the dynamic security constraint $\mathcal{O}(\_, \_)$, see the formulation of semantic soundness (Theorem 5.8).

4.1 Properties of The Type System

For later use we shall list a few basic results. most of which are quite standard. We say that Δ agrees with Δ' on X, to be written $\Delta \underset{X}{=} \Delta'$, if $\Delta(\xi) = \Delta'(\xi)$ for all tags ξ occurring in the entity X.

Lemma 4.3. *Suppose that $\Delta, E \vdash_g P : b$, and that $\Delta \underset{P}{=} \Delta'$. Then also $\Delta', E \vdash_g P : b$.* □

Lemma 4.4 (Swapping)**.** *Assume that $E_1, n_1 : \sigma_1, n_2 : \sigma_2, E_2 \vdash M : \tau$ with $n_1 \neq n_2$. Then it follows that $E_1, n_2 : \sigma_2, n_1 : \sigma_1, E_2 \vdash M : \tau$, with a derivation of the same shape.*
Similarly for $\Delta, E_1, n_1 : \sigma_1, n_2 : \sigma_2, E_2 \vdash_g P : b$. □

Proof. By induction on derivations. □

Lemma 4.5 (Weakening)**.** *Assume that $\Delta, E \vdash_g P : b$, and that n is a name not in $\mathsf{names}(P)$. Then for all τ it follows that $\Delta, E, n : \tau \vdash_g P : b$, with a derivation of the same shape. Similarly for a judgment $E \vdash M : \tau'$ (with $n \notin \mathsf{names}(M)$).* □

Proof. The proof is by induction on derivations, applying Lemma 4.4 to deal with the cases (Proc Res) and (Proc Input). □

Non-structural Rules

(Proc Subsumption)

$$\frac{\Delta, E \vdash_g P : b}{\Delta, E \vdash_g P : b'} \quad (b \leqslant b')$$

(Exp Subsumption)

$$\frac{E \vdash M : \tau}{E \vdash M : \tau'} \quad (\tau \leqslant \tau')$$

Expressions

(Exp n)

$$\frac{E(n) = \tau}{E \vdash n : \tau}$$

(Exp ϵ)

$$\frac{}{E \vdash \epsilon : \mathsf{cap}[\Box]}$$

(Exp Action)

$$\frac{E \vdash M_1 : \mathsf{cap}[B_1] \quad E \vdash M_2 : \mathsf{cap}[B_2]}{E \vdash M_1.M_2 : \mathsf{cap}[B_1 \lfloor B_2 \rfloor]}$$

(Exp In)

$$\frac{E \vdash M : \mathsf{amb}^g_H[br]}{E \vdash \mathsf{in}\ M : \mathsf{cap}[\,^H\mathsf{enter}(g).\Box]}$$

(Exp Out)

$$\frac{E \vdash M : \mathsf{amb}^g_H[br]}{E \vdash \mathsf{out}\ M : \mathsf{cap}[\,^g\mathsf{exit}(H).\Box]}$$

(Exp Open)

$$\frac{E \vdash M : \mathsf{amb}^g_H[\{g_i : b_i\}_{i \in I}]}{E \vdash \mathsf{open}\ M : \mathsf{cap}[_G\mathsf{open}(g).(b \mid \Box)]} \quad \text{if} \quad \begin{array}{l} \forall g' \in G : \exists i \in I : \\ g' = g_i \text{ and } b_i \leqslant b \end{array}$$

Processes

(Proc Zero)

$$\frac{}{\Delta, E \vdash_g \mathbf{0} : \varepsilon}$$

(Proc Par)

$$\frac{\Delta, E \vdash_g P_1 : b_1 \qquad \Delta, E \vdash_g P_2 : b_2}{\Delta, E \vdash_g P_1 \mid P_2 : b_1 \mid b_2}$$

(Proc Repl)

$$\frac{\Delta, E \vdash_g P : b}{\Delta, E \vdash_g !P : b} \quad \text{if } (b \mid b) \leqslant b$$

(Proc Res)

$$\frac{\Delta, E, n : \mathsf{amb}^{g_0}_H[br] \vdash_g P : b}{\Delta, E \vdash_g (\nu n : \mathsf{amb}^{g_0}_H[br]).P : b}$$

(Proc Action)

$$\frac{E \vdash M : \mathsf{cap}[B] \qquad \Delta, E \vdash_g P : b}{\Delta, E \vdash_g M.P : B \lfloor b \rfloor}$$

(Proc Amb)

$$\frac{E \vdash M : \mathsf{amb}^{g_0}_H[br] \quad \Delta, E \vdash_{g_0} P : b}{\Delta, E \vdash_g M[P]^\xi : \varepsilon} \quad \text{if} \quad \begin{array}{l} (g^\uparrow, b) \overset{g_0}{\rightsquigarrow} (H, br) \\ \mathit{group}(\xi) = g_0 \\ \Delta(\xi) = b \end{array}$$

(Proc Input)

$$\frac{\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_g P : b}{\Delta, E \vdash_g (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P : \mathsf{get}(\sigma).b}$$

(Proc Output)

$$\frac{\forall i \in \{1 \ldots k\} : E \vdash M_i : \tau_i}{\Delta, E \vdash_g \langle M_1 \ldots M_k \rangle : \mathsf{put}(\sigma)}$$

In (Proc Input) and (Proc Output), $\sigma = \times(\tau_1, \ldots, \tau_k)$ and $k \geqslant 0$.

Figure 7. Typing Rules.

Lemma 4.6 (Strengthening)**.** *Assume that* $\Delta, E, n : \tau \vdash_g P : b$*, with* n *not in* $\mathsf{names}(P)$*. Then also* $\Delta, E \vdash_g P : b$ *holds, and with a derivation of the same shape. Similarly for* $E, n : \tau \vdash M : \tau'$ *with* $n \notin \mathsf{names}(M)$*.* □

Proof. Similar to the proof of Lemma 4.5. □

5 Subject Reduction

We shall now show that our type system is semantically sound. This property is formulated as a subject reduction result (Theorem 5.8), intuitively stating that "well-typed processes behave according to their behavior" and also stating that "well-typed processes never evolve into ill-typed processes".

The proof of this theorem, to be given shortly, makes heavy use of Lemmas 3.24 and 3.25. Additionally, we shall need several auxiliary results (and one assumption) as listed below.

To deal with the case (Red Open), we need

Lemma 5.1. *Suppose that* $\Delta, E \vdash_g P : b$. *If* $\mathcal{O}(g, g')$ *then also* $\Delta, E \vdash_{g'} P : b$. □

Proof. Structural induction on the derivation, where the only non-trivial case is (Proc Amb). Noting that $g' \in g^{\uparrow}$ and therefore $g'^{\uparrow} \subseteq g^{\uparrow}$, Lemma 3.24 will ensure that the side condition still holds. □

To deal with the case (Red Comm), we need a standard substitution lemma (to be proved in Appendix A):

Lemma 5.2 (Substitution)**.** *Assume that* $E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash M : \tau$ *(with* $n_1 \ldots n_k$ *distinct), and that there exists* $V_1 \ldots V_k$ *such that for all* $i \in \{1 \ldots k\}$ *it holds that* $E \vdash V_i : \tau_i$ *and that* M *is non-conflicting with* $\{n_i\} \cup \mathsf{names}(V_i)$. *Then* $E \vdash M[n_i := V_i] : \tau$. *Similarly, if* $\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_g P : b$ *and for all* i *it holds that* $E \vdash V_i : \tau_i$ *and that* P *is non-conflicting with* $\{n_i\} \cup \mathsf{names}(V_i)$ *then* $\Delta, E \vdash_g P[n_i := V_i] : b$. □

To deal with the case (Red MctxtP), we need the following assumption on the reduction relation on expressions:

Assumption 5.3 (Subject reduction for expressions)**.** Suppose $M_1 \longrightarrow M_2$. If $E \vdash M_1 : \tau$ then also $E \vdash M_2 : \tau$. □

To deal with the case (Red PctxtP), we need the following result (to be proved in Appendix A):

Lemma 5.4 (Reduction of subprocess)**.** *Assume that*

$$\Delta, E \vdash_g \mathcal{PC}\lfloor P \rfloor_p : b.$$

Then there exists E_0, g_0, *and* b_0 *such that*

$$\Delta, E_0 \vdash_{g_0} P : b_0$$

and with the property that if there also exists a judgement

$$\Delta', E_0 \vdash_{g_0} Q : b_0$$

where $\Delta \underset{\mathcal{PC}}{=\!=} \Delta'$, *then also*

$$\Delta', E \vdash_g \mathcal{PC}\lfloor Q \rfloor_p : b.$$ □

To deal with the case (Red $\equiv$), we need to establish (as done in Appendix A):

Lemma 5.5 (Subject congruence)**.** *Assume that* $P \equiv Q$.
Then $\Delta, E \vdash_g P : b$ *if and only if* $\Delta, E \vdash_g Q : b$. □

We are now almost ready to state the soundness result. In order to do it in a succinct way, we introduce a relation $\Delta \xrightarrow{\ell} \Delta'$, expressing that the behaviors in Δ evolves into Δ' as predicted by the reduction label ℓ. Formally, we have

Definition 5.6. The relation $\Delta \xrightarrow{\ell} \Delta'$ holds iff

- Δ' agrees with Δ on $\mathrm{dom}(\Delta) \setminus dom(\ell)$, and
- if $\ell = \xi : \mathsf{enter}\ \chi$ then ${}^{\emptyset}\mathsf{enter}(group(\chi)).\Delta'(\xi) \leqslant \Delta(\xi)$;
- if $\ell = \xi : \mathsf{exit}\ \chi$ then ${}^{group(\chi)}\mathsf{exit}(\emptyset).\Delta'(\xi) \leqslant \Delta(\xi)$;
- if $\ell = \xi : \mathsf{open}\ \chi$ then $\mathsf{Grps}\mathsf{open}(group(\chi)).\Delta'(\xi) \leqslant \Delta(\xi)$;
- if $\ell = \xi : \mathsf{comm}\ \sigma$ then $\exists \sigma' \leqslant \sigma\colon \mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi) \leqslant \Delta(\xi)$.

Here the function $dom(\ell)$ is given by stipulating $dom(\epsilon) = \emptyset$; $dom(\xi : \mathsf{enter}\ \chi) = \{\xi\}$; $dom(\xi : \mathsf{exit}\ \chi) = \{\xi\}$; $dom(\xi : \mathsf{open}\ \chi) = \{\xi\}$; $dom(\xi : \mathsf{comm}\ \sigma) = \{\xi\}$. □

EXAMPLE 5.7. If $\ell = \xi : \mathsf{enter}\ \chi$ and Δ maps ξ into ${}^{A}\mathsf{enter}(B).{}^{B}\mathsf{exit}(A)$, then $\Delta \xrightarrow{\ell} \Delta'$ holds if $group(\chi) = B$ (the ambient tagged ξ is in fact steered into an ambient with group B) and Δ' maps ξ into ${}^{B}\mathsf{exit}(A)$ (the remaining action). □

Theorem 5.8 (Subject reduction for processes)**.** *Suppose that*

$$P \xrightarrow{\ell} Q$$

where P and Q are uniquely tagged, and where all tags occurring in Q but not in P do not occur in $\mathrm{dom}(\Delta)$. *Further assume that*

$$\Delta, E \vdash_g P : b.$$

Then there exists Δ' such that

$$\Delta \xrightarrow{\ell} \Delta'$$
$$\Delta', E \vdash_g Q : b$$

*and additionally (***Safety of opening***): if $\ell = \xi : \mathsf{open}\ \chi$ then $\mathcal{O}(group(\chi), group(\xi))$.* □

Proof. The proof is by induction on the derivation of $P \xrightarrow{\ell} Q$, with a case analysis on the last rule used. In the type derivations show below, we shall "inline" the applications of the subtyping rules (Proc Subsumption) and (Exp Subsumption).

(Red In). The situation is that

$$m[\mathsf{in}\ n.P \mid Q]^{\xi} \mid n[R]^{\chi} \xrightarrow{\xi:\mathsf{enter}\ \chi} n[m[P \mid Q]^{\xi} \mid R]^{\chi}$$

With $\Delta(\xi) = b_m$ and $\Delta(\chi) = b_n$, and with $group(\xi) = g_m$ and $group(\chi) = g_n$ (note that $\xi \neq \chi$ by assumption on unique tags), the typing of the left hand side takes the form

$$\dfrac{\dfrac{E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m] \qquad \dfrac{\dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H'_n}[br'_n]}{E \vdash \mathsf{in}\ n : \mathsf{cap}[B]} \quad \Delta, E \vdash_{g_m} P : b_0}{\Delta, E \vdash_{g_m} \mathsf{in}\ n.P : b_1} \qquad \Delta, E \vdash_{g_m} Q : b_2}{\Delta, E \vdash_{g_m} \mathsf{in}\ n.P \mid Q : b_m}}{\Delta, E \vdash_g m[\mathsf{in}\ n.P \mid Q]^{\xi} : \varepsilon} \qquad \dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H_n}[br_n] \quad \Delta, E \vdash_{g_n} R : b_n}{\Delta, E \vdash_g n[R]^{\chi} : \varepsilon}}{\Delta, E \vdash_g m[\mathsf{in}\ n.P \mid Q]^{\xi} \mid n[R]^{\chi} : b}$$

where we have the relations

$$^{H'_n}\mathsf{enter}(g_n).\square \leqslant B \qquad B\lfloor b_0 \rfloor \leqslant b_1 \qquad b_1 \mid b_2 \leqslant b_m \qquad \varepsilon \leqslant b$$

$$(g^{\uparrow}, b_m) \overset{g_m}{\rightsquigarrow} (H_m, br_m) \tag{1}$$

$$(g^{\uparrow}, b_n) \overset{g_n}{\rightsquigarrow} (H_n, br_n) \tag{2}$$

We now define

$$\Delta' = \Delta[\xi \mapsto (b_0 \mid b_2)]$$

By Lemma 3.12 (cf. the remark after Def. 3.4), it holds that

$$^{H'_n}\mathsf{enter}(g_n).b_0 \leqslant B\lfloor b_0 \rfloor \leqslant b_1.$$

From (2) we see by Lemma 3.23 that $g^{\uparrow} \subseteq H_n$, which since $\mathsf{amb}^{\_}_{H}[\_]$ is *invariant* in H and therefore $H_n \subseteq H'_n$ amounts to

$$g^{\uparrow} \subseteq H'_n.$$

By Lemma 3.8 we therefore infer that

$$^{g^{\uparrow}}\mathsf{enter}(g_n).(b_0 \mid b_2) \leqslant (\,^{g^{\uparrow}}\mathsf{enter}(g_n).b_0) \mid b_2 \leqslant (\,^{H'_n}\mathsf{enter}(g_n).b_0) \mid b_2 \leqslant b_1 \mid b_2 \leqslant b_m. \tag{3}$$

By Lemma 3.24 applied to (1), this shows

$$(g^{\uparrow}, \,^{g^{\uparrow}}\mathsf{enter}(g_n).(b_0 \mid b_2)) \overset{g_m}{\rightsquigarrow} (H_m, br_m)$$

which by Lemma 3.25 implies that

$$(g_n{}^{\uparrow}, b_0 \mid b_2) \overset{g_m}{\rightsquigarrow} (H_m, br_m). \tag{4}$$

Since the left hand side of the reduction is uniquely tagged, ξ does not occur in either of P, Q, or R. Thus $\Delta \overset{\xi}{\underset{P,Q,R}{=}} \Delta'$, so by Lemma 4.3 it holds that

$$\Delta', E \vdash_{g_m} P : b_0 \text{ and } \Delta', E \vdash_{g_m} Q : b_2 \text{ and } \Delta', E \vdash_{g_n} R : b_n.$$

Using the above observations, in particular (4), we can indeed construct a typing for the right hand side of the reduction:

$$\dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H_n}[br_n] \qquad \dfrac{E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m] \qquad \dfrac{\Delta', E \vdash_{g_m} P : b_0 \qquad \Delta', E \vdash_{g_m} Q : b_2}{\Delta', E \vdash_{g_m} P \mid Q : b_0 \mid b_2}}{\Delta', E \vdash_{g_n} m[P \mid Q]^{\xi} : \varepsilon} \qquad \Delta', E \vdash_{g_n} R : b_n}{\Delta', E \vdash_{g_n} m[P \mid Q]^{\xi} \mid R : b_n}$$

$$\Delta', E \vdash_g n[m[P \mid Q]^{\xi} \mid R]^{\chi} : b$$

Moreover, by (3) we also have the desired relation

$$^{\emptyset}\mathsf{enter}(group(\chi)).\Delta'(\xi) = {}^{\emptyset}\mathsf{enter}(g_n).(b_0 \mid b_2) \leqslant b_m = \Delta(\xi).$$

(Red Out). The situation is that

$$n[m[\mathsf{out}\ n.P \mid Q]^{\xi} \mid R]^{\chi} \xrightarrow{\xi:\mathsf{exit}\ \chi} m[P \mid Q]^{\xi} \mid n[R]^{\chi}$$

With $\Delta(\xi) = b_m$ and $\Delta(\chi) = b_n$, and with $group(\xi) = g_m$ and $group(\chi) = g_n$ (note that $\xi \neq \chi$ by assumption on unique tags), the typing of the left hand side takes the form

$$\dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H_n}[br_n] \qquad \dfrac{\dfrac{\dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H'_n}[br'_n]}{E \vdash \mathsf{out}\ n : \mathsf{cap}[B] \quad \Delta, E \vdash_{g_m} P : b_0} \qquad \dfrac{\Delta, E \vdash_{g_m} \mathsf{out}\ n.P : b_1 \qquad \Delta, E \vdash_{g_m} Q : b_2}{\Delta, E \vdash_{g_m} \mathsf{out}\ n.P \mid Q : b_m}}{\Delta, E \vdash_{g_n} m[\mathsf{out}\ n.P \mid Q]^{\xi} : \varepsilon} \qquad \Delta, E \vdash_{g_n} R : b_n}{\Delta, E \vdash_{g_n} m[\mathsf{out}\ n.P \mid Q]^{\xi} \mid R : b_n}}{\Delta, E \vdash_{g} n[m[\mathsf{out}\ n.P \mid Q]^{\xi} \mid R]^{\chi} : b}$$

where we have the relations

$$^{g_n}\mathsf{exit}(H'_n).\square \leqslant B \qquad B\lfloor b_0 \rfloor \leqslant b_1 \qquad b_1 \mid b_2 \leqslant b_m \qquad \varepsilon \leqslant b$$

$$(g_n{}^{\uparrow}, b_m) \overset{g_m}{\rightsquigarrow} (H_m, br_m) \tag{5}$$

$$(g^{\uparrow}, b_n) \overset{g_n}{\rightsquigarrow} (H_n, br_n) \tag{6}$$

We now define

$$\Delta' = \Delta[\xi \mapsto (b_0 \mid b_2)]$$

By Lemma 3.12 (cf. the remark after Def. 3.4), it holds that

$$^{g_n}\mathsf{exit}(H'_n).b_0 \leqslant B\lfloor b_0 \rfloor \leqslant b_1.$$

From (6) we see by Lemma 3.23 that $g^{\uparrow} \subseteq H_n$, which since $\mathsf{amb}^{-}_{H}[\_]$ is *invariant* in H and therefore $H_n \subseteq H'_n$ amounts to

$$g^{\uparrow} \subseteq H'_n.$$

By Lemma 3.8 we therefore infer that

$$^{g_n}\mathsf{exit}(g^{\uparrow}).(b_0 \mid b_2) \leqslant (\,^{g_n}\mathsf{exit}(g^{\uparrow}).b_0) \mid b_2 \leqslant (\,^{g_n}\mathsf{exit}(H'_n).b_0) \mid b_2 \leqslant b_1 \mid b_2 \leqslant b_m. \tag{7}$$

By Lemma 3.24 applied to (5), this shows

$$(g_n{}^{\uparrow},\ ^{g_n}\mathsf{exit}(g^{\uparrow}).(b_0 \mid b_2)) \overset{g_m}{\rightsquigarrow} (H_m, br_m)$$

which by Lemma 3.25 implies that

$$(g^{\uparrow}, b_0 \mid b_2) \overset{g_m}{\rightsquigarrow} (H_m, br_m). \tag{8}$$

Since the left hand side of the reduction is uniquely tagged, ξ does not occur in neither of P, Q, or R. Therefore (by Lemma 4.3) it holds that

$\Delta', E \vdash_{g_m} P : b_0$ and $\Delta', E \vdash_{g_m} Q : b_2$ and $\Delta', E \vdash_{g_n} R : b_n$

Using the above observations, in particular (8), we can indeed construct a typing for the right hand side of the reduction:

$$\dfrac{\dfrac{E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m] \qquad \dfrac{\Delta', E \vdash_{g_m} P : b_0 \qquad \Delta', E \vdash_{g_m} Q : b_2}{\Delta', E \vdash_{g_m} P \mid Q : b_0 \mid b_2}}{\Delta', E \vdash_g m[\![P \mid Q]\!]^{\xi} : \varepsilon} \qquad \dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H_n}[br_n] \qquad \Delta', E \vdash_{g_n} R : b_n}{\Delta', E \vdash_g n[\![R]\!]^{\chi} : \varepsilon}}{\Delta', E \vdash_g m[\![P \mid Q]\!]^{\xi} \mid n[\![R]\!]^{\chi} : b}$$

Moreover, by (7) we also have the desired relation

$${}^{group(\chi)}\mathsf{exit}(\emptyset).\Delta'(\xi) = {}^{g_n}\mathsf{exit}(\emptyset).(b_0 \mid b_2) \leqslant b_m = \Delta(\xi).$$

(Red Open). The situation is that

$$m[\![\mathsf{open}\ n.P \mid n[\![Q]\!]^{\chi} \mid R]\!]^{\xi} \xrightarrow{\xi:\mathsf{open}\ \chi} m[\![P \mid Q \mid R]\!]^{\xi}$$

With $\Delta(\xi) = b_m$ and $\Delta(\chi) = b_n$, and with $group(\xi) = g_m$ and $group(\chi) = g_n$ (note that $\xi \neq \chi$ by assumption on unique tags), the typing of the left hand side takes the form

$$\dfrac{E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m] \qquad \dfrac{\dfrac{\dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H'_n}[br'_n]}{E \vdash \mathsf{open}\ n : \mathsf{cap}[B]} \quad \Delta, E \vdash_{g_m} P : b_0}{\Delta, E \vdash_{g_m} \mathsf{open}\ n.P : b_1} \qquad \dfrac{E \vdash n : \mathsf{amb}^{g_n}_{H_n}[br_n] \quad \Delta, E \vdash_{g_n} Q : b_n}{\Delta, E \vdash_{g_m} n[\![Q]\!]^{\chi} : \varepsilon} \qquad \Delta, E \vdash_{g_m} R : b_2}{\Delta, E \vdash_{g_m} \mathsf{open}\ n.P \mid n[\![Q]\!]^{\chi} \mid R : b_m}}{\Delta, E \vdash_g m[\![\mathsf{open}\ n.P \mid n[\![Q]\!]^{\chi} \mid R]\!]^{\xi} : b}$$

where we have the relations

$${}_G\mathsf{open}(g_n).(b' \mid \square) \leqslant B \qquad B\lfloor b_0 \rfloor \leqslant b_1 \qquad b_1 \mid b_2 \leqslant b_m \qquad \varepsilon \leqslant b$$

$$(g_m{}^{\uparrow}, b_n) \overset{g_n}{\rightsquigarrow} (H_n, br_n) \tag{9}$$

$$(g^{\uparrow}, b_m) \overset{g_m}{\rightsquigarrow} (H_m, br_m) \tag{10}$$

With $br'_n = \{g'_i : b'_i\}_{i \in I}$ we also have

$$\forall g' \in G : \exists i \in I : g' = g'_i \text{ and } b'_i \leqslant b' \tag{11}$$

Since $\mathsf{amb}\text{-}[br]$ is *invariant* in br, we have $br_n \leqslant br'_n$ and $br'_n \leqslant br_n$, so (9) clearly implies

$$(g_m{}^{\uparrow}, b_n) \overset{g_n}{\rightsquigarrow} (H_n, br'_n) \tag{12}$$

We now define

$$\Delta' = \Delta[\xi \mapsto (b_0 \mid b_n \mid b_2)]$$

By Lemma 3.12 (cf. the remark after Def. 3.4), it holds that

$${}_G\mathsf{open}(g_n).(b' \mid b_0) \leqslant B\lfloor b_0 \rfloor \leqslant b_1.$$

By Lemma 3.8 we therefore infer that

$$ {}_G\mathsf{open}(g_n).(b' \mid b_0 \mid b_2) \leqslant ({}_G\mathsf{open}(g_n).(b' \mid b_0)) \mid b_2 \leqslant b_1 \mid b_2 \leqslant b_m. \tag{13} $$

By Lemma 3.24 applied to (10), this shows

$$ (g^\uparrow, {}_G\mathsf{open}(g_n).(b' \mid b_0 \mid b_2)) \overset{g_m}{\rightsquigarrow} (H_m, br_m) $$

As ${}_G\mathsf{open}(g_n).(b' \mid b_0 \mid b_2)$ contains a trace (we exploit that all behaviors are non-empty) of the form $\bullet \diamond {}_G\mathsf{open}(g_n) \diamond tr_3$ with $\bullet$ feasible from $g^\uparrow$, Definition 3.19 tells us that $g_m \in G$. By (11) there thus exists $i \in I$ such that

$$ g_m = g'_i \text{ and } b'_i \leqslant b'. $$

From this we infer, since $\mathsf{amb}^{g_n}_{H'_n}[br'_n]$ is a (well-formed) type, that

$$ \mathcal{O}(g_n, g_m) \tag{14} $$

and also infer that $g'_i \in {g_m}^\uparrow$. Lemma 3.23 applied to (12) then tells us that $b_n \leqslant b'_i$ and therefore

$$ \Delta'(\xi) = b_0 \mid b_n \mid b_2 \leqslant b' \mid b_0 \mid b_2. $$

From (13) we thus infer

$$ {}_G\mathsf{open}(g_n).\Delta'(\xi) \leqslant b_m \tag{15} $$

which by Lemma 3.24 applied to (10) shows $(g^\uparrow, {}_G\mathsf{open}(g_n).\Delta'(\xi)) \overset{g_m}{\rightsquigarrow} (H_m, br_m)$. By Lemma 3.25 this implies that

$$ (g^\uparrow, \Delta'(\xi)) \overset{g_m}{\rightsquigarrow} (H_m, br_m). \tag{16} $$

Since the left hand side of the reduction is uniquely tagged, ξ does not occur in neither of P, Q, or R. Therefore (by Lemma 4.3) it holds that

$$ \Delta', E \vdash_{g_m} P : b_0 \text{ and } \Delta', E \vdash_{g_m} R : b_2 $$

as well as $\Delta', E \vdash_{g_n} Q : b_n$ which by Lemma 5.1 using (14) implies

$$ \Delta', E \vdash_{g_m} Q : b_n. $$

Using the above observations, in particular (16), we can indeed construct a typing for the right hand side of the reduction:

$$ \dfrac{E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m] \qquad \dfrac{\Delta', E \vdash_{g_m} P : b_0 \qquad \Delta', E \vdash_{g_m} Q : b_n \qquad \Delta', E \vdash_{g_m} R : b_2}{\Delta', E \vdash_{g_m} P \mid Q \mid R : \Delta'(\xi)}}{\Delta', E \vdash_g m[\![P \mid Q \mid R]\!]^\xi : b} $$

Moreover, by (15) we also (employing the polarity of $\_\mathsf{open}(\_)$) have the desired relation

$$ {}_{\mathsf{Grps}}\mathsf{open}(group(\chi)).\Delta'(\xi) \leqslant {}_G\mathsf{open}(g_n).\Delta'(\xi) \leqslant b_m = \Delta(\xi) $$

and by (14) also the desired safety predicate

$$ \mathcal{O}(group(\chi), group(\xi)). $$

(Red Comm). The situation is that

$$m[\![(n_1 \dots n_k : \tau_1 \dots \tau_k).P \mid \langle V_1 \dots V_k \rangle \mid Q]\!]^\xi \xrightarrow{\xi:\mathsf{comm} \times(\tau_1,\dots,\tau_k)} m[\![P[n_i := V_i] \mid Q]\!]^\xi$$

With $\Delta(\xi) = b_m$ and $group(\xi) = g_m$, the typing of the left hand side takes the form

$$\dfrac{\dfrac{\dfrac{\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_{g_m} P : b_0}{\Delta, E \vdash_{g_m} (n_1 \dots n_k : \tau_1 \dots \tau_k).P : b_1} \quad \dfrac{\cdots E \vdash V_i : \tau_i' \cdots}{\Delta, E \vdash_{g_m} \langle V_1 \dots V_k \rangle : b_2} \quad \Delta, E \vdash_{g_m} Q : b_3}{\Delta, E \vdash_{g_m} (n_1 \dots n_k : \tau_1 \dots \tau_k).P \mid \langle V_1 \dots V_k \rangle \mid Q : b_m} \quad E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m]}{\Delta, E \vdash_g m[\![(n_1 \dots n_k : \tau_1 \dots \tau_k).P \mid \langle V_1 \dots V_k \rangle \mid Q]\!]^\xi : b}$$

where with $\sigma = \times(\tau_1, \dots, \tau_k)$ and $\sigma' = \times(\tau_1', \dots, \tau_k')$ we have the relations

$$\mathsf{get}(\sigma).b_0 \leqslant b_1 \qquad \mathsf{put}(\sigma') \leqslant b_2 \qquad b_1 \mid b_2 \mid b_3 \leqslant b_m \qquad \varepsilon \leqslant b$$
$$(g^\uparrow, b_m) \overset{g_m}{\rightsquigarrow} (H_m, br_m) \tag{17}$$

We now define

$$\Delta' = \Delta[\xi \mapsto (b_0 \mid b_3)]$$

By Lemma 3.8 we infer that

$$\begin{array}{l}\mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi) \leqslant \mathsf{put}(\sigma').(\mathsf{get}(\sigma).b_0 \mid b_3) \leqslant \mathsf{put}(\sigma').(\varepsilon \mid (b_1 \mid b_3)) \leqslant \\ (\mathsf{put}(\sigma').\varepsilon) \mid (b_1 \mid b_3) \leqslant b_2 \mid b_1 \mid b_3 \leqslant b_m\end{array} \tag{18}$$

which by Lemma 3.24 applied to (17) shows

$$(g^\uparrow, \mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi)) \overset{g_m}{\rightsquigarrow} (H_m, br_m) \tag{19}$$

As $\mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi)$ contains a trace (we exploit that all behaviors are non-empty) of the form $\bullet \diamond \mathsf{put}(\sigma') \diamond \mathsf{get}(\sigma) \diamond tr_3$ with $\bullet$ feasible from $g^\uparrow$, Definition 3.19 tells us that

$$\sigma' \leqslant \sigma \text{ and thus } \forall i \in \{1 \dots k\} : \tau_i' \leqslant \tau_i. \tag{20}$$

Lemma 3.25 applied to (19) tells us that

$$(g^\uparrow, \Delta'(\xi)) \overset{g_m}{\rightsquigarrow} (H_m, br_m). \tag{21}$$

For all $i \in \{1 \dots k\}$ we from (20) and $E \vdash V_i : \tau_i'$ infer that

$$E \vdash V_i : \tau_i.$$

By assumption, $(n_1 \dots n_k : \sigma_1 \dots \sigma_k).P$ is non-conflicting with $\mathsf{names}(V_i)$, and therefore P is non-conflicting with $\{n_i\} \cup \mathsf{names}(V_i)$. We can thus apply Lemma 5.2 to the judgement

$$\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_{g_m} P : b_0$$

to infer that

$$\Delta, E \vdash_{g_m} P[n_i := V_i] : b_0.$$

Since the left hand side of the reduction is uniquely tagged, ξ does not occur in neither of P or Q (and certainly not in any of the V_i's). Therefore (by Lemma 4.3) it holds that

$\Delta', E \vdash_{g_m} P[n_i := V_i] : b_0$ and $\Delta', E \vdash_{g_m} Q : b_3$.

Using the above observations, in particular (21), we can indeed construct a typing for the right hand side of the reduction:

$$\dfrac{E \vdash m : \mathsf{amb}^{g_m}_{H_m}[br_m] \qquad \dfrac{\Delta', E \vdash_{g_m} P[n_i := V_i] : b_0 \qquad \Delta', E \vdash_{g_m} Q : b_3}{\Delta', E \vdash_{g_m} P[n_i := V_i] \mid Q : \Delta'(\xi)}}{\Delta', E \vdash_g m[\![P[n_i := V_i] \mid Q]\!]^{\xi} : b}$$

Moreover, by (18) and (20), we also have the desired relation

$\mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi) \leqslant b_m = \Delta(\xi)$ where $\sigma' \leqslant \sigma$.

(Red Repl). The situation is that

$$!P \xrightarrow{\epsilon} P' \mid !P$$

where $P' \equiv_{\mathrm{t}} P$ (cf. Definition 2.1). The typing of the left hand side takes the form

$$\dfrac{\dfrac{\Delta, E \vdash_g P : b_0}{\Delta, E \vdash_g !P : b_0}}{\Delta, E \vdash_g !P : b}$$

where $b_0 \mid b_0 \leqslant b_0$ and $b_0 \leqslant b$. We now define Δ' as follows:

- $\Delta'(\xi) = \Delta(\xi)$ if ξ occurs in $\mathrm{dom}(\Delta)$;
- $\Delta'(\xi) = \Delta(\chi)$ if ξ occurs in P' with χ the tag of the corresponding position in P.

Using our assumptions, it is easy to see that this is well-defined. Moreover, since $\Delta, E \vdash_g P : b_0$ it clearly holds that

$$\Delta', E \vdash_g P' : b_0$$

and (by Lemma 4.3) also

$$\Delta', E \vdash_g P : b_0.$$

Using the above observations, in particular that $b_0 \mid b_0 \leqslant b_0$, we can indeed construct a typing for the right hand side of the reduction:

$$\dfrac{\dfrac{\Delta', E \vdash_g P' : b_0 \quad \dfrac{\Delta', E \vdash_g P : b_0}{\Delta', E \vdash_g !P : b_0}}{\Delta', E \vdash_g P' \mid !P : b_0}}{\Delta', E \vdash_g P' \mid !P : b}$$

(Red MctxtP). The situation is that

$$\mathcal{PC}\lfloor M_1 \rfloor_e \xrightarrow{\epsilon} \mathcal{PC}\lfloor M_2 \rfloor_e$$

because $M_1 \longrightarrow M_2$, and that

$$\Delta, E \vdash_g \mathcal{PC}\lfloor M_1 \rfloor_e : b.$$

Clearly there exists E_1 and τ such that $E_1 \vdash M_1 : \tau$. By Assumption 5.3, this implies $E_1 \vdash M_2 : \tau$. It is easy to see that then also $\Delta, E \vdash_g \mathcal{PC}\lfloor M_2 \rfloor_e : b$. The claim thus follows, with $\Delta' = \Delta$.

(Red PctxtP). The situation is that

$$\mathcal{PC}\lfloor P \rfloor_p \xrightarrow{\ell} \mathcal{PC}\lfloor Q \rfloor_p$$

because $P \xrightarrow{\ell} Q$, and that

$$\Delta, E \vdash_g \mathcal{PC}\lfloor P \rfloor_p : b. \tag{22}$$

By Lemma 5.4 there exists E_0, g_0, and b_0 such that

$$\Delta, E_0 \vdash_{g_0} P : b_0.$$

We can thus apply the induction hypothesis on $P \xrightarrow{\ell} Q$ to find Δ' such that

$$\Delta', E_0 \vdash_{g_0} Q : b_0$$

and such that Δ' has certain properties dependent on ℓ, in particular

$$\Delta' \text{ agrees with } \Delta \text{ on } \mathrm{dom}(\Delta) \setminus dom(\ell). \tag{23}$$

We can thus apply Lemma 5.4 to arrive at the desired judgement

$$\Delta', E \vdash_g \mathcal{PC}\lfloor Q \rfloor_p : b$$

provided we can show that $\Delta \underset{\overline{\mathcal{PC}}}{=} \Delta'$.

So let ξ be a tag occurring in $\mathcal{PC}$; we must show $\Delta(\xi) = \Delta'(\xi)$. Thus ξ occurs in $\mathcal{PC}\lfloor P \rfloor_p$ and therefore (from (22)) clearly $\xi \in \mathrm{dom}(\Delta)$, and (since $\mathcal{PC}\lfloor P \rfloor_p$ is uniquely tagged by assumption) ξ does not occur in P which by Lemma 2.4 (applied to $P \xrightarrow{\ell} Q$) shows that ξ does not occur in $dom(\ell)$. Thus $\xi \in \mathrm{dom}(\Delta) \setminus dom(\ell)$. But by (23) this shows that $\Delta'(\xi) = \Delta(\xi)$, as desired.

(Red $\equiv$). The situation is that

$$P' \xrightarrow{\ell} Q'$$

because $P' \equiv P$ and $P \xrightarrow{\ell} Q$ and $Q \equiv Q'$, and that

$$\Delta, E \vdash_g P' : b.$$

By Lemma 5.5 we infer that

$$\Delta, E \vdash_g P : b.$$

We can therefore apply the induction hypothesis (thanks to Lemma 2.3) to find Δ' such that

$$\Delta', E \vdash_g Q : b$$

and such that Δ' has certain properties dependent on ℓ. This is as desired, since by one more application of Lemma 5.5 we arrive at

$$\Delta', E \vdash_g Q' : b.$$

□

6 Type Checking

In this section we show that given a complete type derivation for some process P, we can check its validity according to the rules from Fig. 7. For this purpose we need to show: *(i)* that we can decide the side-conditions for all the rules in Fig. 7, and *(ii)* that we can check if two behaviors (types) are identical. Recall that a behavior is a regular set of traces; we shall assume that behaviors are represented by finite automata. Note that two behaviors may be represented by different automata and still be identical.

This section is organized as follows: in Sect. 6.1 we introduce finite automata and some basic constructs on these; in Sect. 6.2 we demonstrate that the preorders $\leqslant$, as well as the equality relation (cf. the remark above), are decidable; in Sect. 6.4 we show, using the notion of annotated automata introduced in Sect. 6.3, that the trace correctness predicate is decidable. Sect. 6.5 puts all these results together.

6.1 Finite Automata

We shall employ non-deterministic finite automata:

Definition 6.1 (Automata)**.** An automaton A is a quadruple $A = (\mathcal{Q}, \iota, F, \delta)$ with

- $\mathcal{Q}$ a finite set of states,
- $\iota \in \mathcal{Q}$ the initial state,
- $F \subseteq \mathcal{Q}$ the set of final states, and
- δ the transition relation: a finite subset of $\mathcal{Q} \times (\mathsf{Act} + \{\epsilon\}) \times \mathcal{Q}$. Here ϵ is a symbol not in Act.

We shall use q to range over states, but also use f to range over final states, use Q to range over set of states, and use t to range over $\mathsf{Act} + \{\epsilon\}$.

We define $\delta^* \subseteq \mathcal{Q} \times \mathsf{Trace} \times \mathcal{Q}$ by stipulating

- $(q, \bullet, q) \in \delta^*$ for all $q \in \mathcal{Q}$;
- $(q, \epsilon, q_1) \in \delta$ and $(q_1, tr, q_2) \in \delta^*$ implies $(q, tr, q_2) \in \delta^*$;
- $(q, a, q_1) \in \delta$ and $(q_1, tr, q_2) \in \delta^*$ implies $(q, a \diamond tr, q_2) \in \delta^*$.

The set of traces accepted by A is defined by

$$\mathsf{Acc}(A) = \{tr \mid \exists f \in F : (\iota, tr, f) \in \delta^*\}$$

We say that a occurs in A if there exists q_1, q_2 such that $(q_1, a, q_2) \in \delta$, and say that A is of level i if all a occurring in A are of level i. □

Lemma 6.2. *If* $(q_1, tr_1, q_2) \in \delta^*$ *and* $(q_2, tr_2, q_3) \in \delta^*$ *then* $(q_1, tr_1 \diamond tr_2, q_3) \in \delta^*$. □

Proof. Induction in the derivation of $(q_1, tr_1, q_2) \in \delta^*$. □

We shall often be interested in automata having a certain form:

Definition 6.3. Let $A = (\mathcal{Q}, \iota, F, \delta)$.

ϵ-free If $\delta \subseteq \mathcal{Q} \times \mathsf{Act} \times \mathcal{Q}$ we say that A is ϵ-free.

junk-free If for all $q \in \mathcal{Q}$ there exists $f \in F$ and $tr \in \mathsf{Trace}$ such that $(q, tr, f) \in \delta^*$, we say that A is junk-free.

normalized If $\iota \notin F$, and δ contains no triple of the form $(f, \_, \_)$ with $f \in F$ nor of the form $(\_, \_, \iota)$, we say that A is normalized. □

Definition 6.4 (Implementation)**.** We say that automaton A implements behavior b if $b = \mathsf{Acc}(A)$. □

By the definition of a behavior as a regular set of traces, for given b there will always be an automaton A implementing it. Using Lemmas 6.5 and 6.6 given below, we can even assume this A to be ϵ-free and junk-free.

Lemma 6.5. *Given automaton A, we can construct ϵ-free A' with $\mathsf{Acc}(A') = \mathsf{Acc}(A)$.* □

Details are given in Appendix A, though the construction is standard: the states of A' are given as the "ϵ-closures" of the states of A. Note that the property of being normalized is not preserved: there may be[10] outgoing transitions from final states in A'.

Lemma 6.6. *Given automaton A with $\mathsf{Acc}(A) \neq \emptyset$, we can construct junk-free A' with $\mathsf{Acc}(A') = \mathsf{Acc}(A)$. Moreover, A' will be ϵ-free if A is ϵ-free, and A' will be normalized if A is normalized.* □

The construction is obvious; note that the condition $\mathsf{Acc}(A) \neq \emptyset$ ensures that the initial state is not "junk".

Lemma 6.7. *Given automaton A, we can construct normalized A' with $\mathsf{Acc}(A') = \mathsf{Acc}(A)$.* □

Proof. Let $A = (\mathcal{Q}, \iota, F, \delta)$. With ι' and q_0 "fresh" states (not in $\mathcal{Q}$), we define $A' = (\mathcal{Q} \cup \{\iota', q_0\}, \iota', \{q_0\}, \delta')$ where $\delta' = \delta \cup \{(\iota', \epsilon, \iota)\} \cup \{(q, \epsilon, q_0) \mid q \in F\}$. This clearly does the job. □

Note that this construction does not preserve the property of being ϵ-free. The following result is immediate:

Lemma 6.8. *Given A, it is decidable whether $\mathsf{Acc}(A) = \emptyset$.* □

Below we shall list some useful constructions on automata, starting with those implementing the operators defined in Sect. 3.

Definition 6.9. The automaton A_ε is given by $A_\varepsilon = (\{q\}, q, \{q\}, \emptyset)$ (with q an arbitrary symbol). □

Lemma 6.10. $\mathsf{Acc}(A_\varepsilon) = \{\bullet\}$. *Moreover, A_ε is ϵ-free and junk-free.* □

Definition 6.11. Given an automaton $A = (\mathcal{Q}, \iota, F, \delta)$ and an action a, the automaton $a.A$ is given by (where ι' is a symbol not in $\mathcal{Q}$)

$$a.A = (\mathcal{Q} \cup \{\iota'\}, \iota', F, \delta \cup (\iota', a, \iota)).$$

□

Lemma 6.12. $\mathsf{Acc}(a.A) = \{a \diamond tr \mid tr \in \mathsf{Acc}(A)\}$. *Moreover, if A is ϵ-free then $a.A$ is; similarly the properties of being junk-free and normalized will be preserved.* □

[10] Let $(q, \epsilon, f) \in \delta$ and $(q, a, q') \in \delta$ with $F = \{f\}$. Then $\epsilon(q)$ will be final, with an a transition into $\epsilon(q')$.

Definition 6.13. Given ϵ-free automata $A_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$ $(j = 1, 2)$, we construct $A_1 \mid A_2 = (\mathcal{Q}, \iota, F, \delta)$ as follows:

$$\begin{array}{rcl} \mathcal{Q} & = & \mathcal{Q}_1 \times \mathcal{Q}_2 \\ \iota & = & (\iota_1, \iota_2) \\ F & = & F_1 \times F_2 \\ \delta & = & \{((q_1, q_2), a, (q_1', q_2)) \mid (q_1, a, q_1') \in \delta_1\} \\ & \cup & \{((q_1, q_2), a, (q_1, q_2')) \mid (q_2, a, q_2') \in \delta_2\} \end{array}$$

□

Lemma 6.14. $\mathsf{Acc}(A_1 \mid A_2) = \mathsf{Acc}(A_1) \mid \mathsf{Acc}(A_2)$. *Moreover, $A_1 \mid A_2$ is ϵ-free, and if A_1 and A_2 are junk-free (resp. normalized) then $A_1 \mid A_2$ is junk-free (resp. normalized).* □

The proof is in Appendix A. By combining the results above, we get

Corollary 6.15. *Assume that b_1 is implemented by the ϵ-free automaton A_1, and that b_2 is implemented by the ϵ-free automaton A_2. Then*

- *$A_1 \mid A_2$ implements $b_1 \mid b_2$;*
- *$a.A_1$ implements $a.b_1$;*
- *A_ε implements ε.* □

The next two constructs are for the purpose of testing $b_1 = b_2$ and $b_1 \leqslant b_2$.

Definition 6.16. Given ϵ-free automata $A_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$ $(j = 1, 2)$, we construct $A_1 \setminus A_2 = (\mathcal{Q}, \iota, F, \delta)$ as follows:

$$\begin{array}{rcl} \mathcal{Q} & = & \mathcal{Q}_1 \times \mathcal{P}(\mathcal{Q}_2) \\ \iota & = & (\iota_1, \{\iota_2\}) \\ F & = & \{(q_1, Q_2) \mid q_1 \in F_1 \text{ and } Q_2 \cap F_2 = \emptyset\} \\ \delta & = & \{((q_1, Q_2), a, (q_1', Q_2')) \mid (q_1, a, q_1') \in \delta_1 \wedge Q_2' = \{q_2' \in \mathcal{Q}_2 \mid \exists q_2 \in Q_2 : (q_2, a, q_2') \in \delta_2\}\} \end{array}$$

Similarly, we construct $A_1 \setminus^{\leq} A_2 = (\mathcal{Q}, \iota, F, \delta)$ as follows:

$$\begin{array}{rcl} \mathcal{Q} & = & \mathcal{Q}_1 \times \mathcal{P}(\mathcal{Q}_2) \\ \iota & = & (\iota_1, \{\iota_2\}) \\ F & = & \{(q_1, Q_2) \mid q_1 \in F_1 \text{ and } Q_2 \cap F_2 = \emptyset\} \\ \delta & = & \{((q_1, Q_2), a, (q_1', Q_2')) \mid (q_1, a, q_1') \in \delta_1 \wedge Q_2' = \{q_2' \in \mathcal{Q}_2 \mid \exists q_2 \in Q_2, a \leqslant a^+ : (q_2, a^+, q_2') \in \delta_2\}\} \end{array}$$

□

The following result is immediate:

Lemma 6.17. *Assume that for all a_1 occurring in A_1 and all a_2 occurring in A_2, the relation $a_1 = a_2$, resp. $a_1 \leqslant a_2$, is decidable. Then $A_1 \setminus A_2$, resp. $A_1 \setminus^{\leq} A_2$, can be effectively constructed. Moreover, these automata are ϵ-free.* □

Lemma 6.18. $\mathsf{Acc}(A_1 \setminus A_2) = \mathsf{Acc}(A_1) \setminus \mathsf{Acc}(A_2)$.
$\mathsf{Acc}(A_1 \setminus^{\leq} A_2) = \{tr \in \mathsf{Acc}(A_1) \mid \mathit{not}(\exists tr^+ \in \mathsf{Acc}(A_2) : tr \leqslant tr^+)\}$. □

The proof is in Appendix A.

6.2 Subtyping is Decidable

Theorem 6.19. *The relations "$\leqslant$" defined in Sect. 3.1 are decidable. Also, the identity relation "$=$" is decidable.* □

Proof. The decision procedures are defined mutually recursively, in a way that is clearly well-defined:

1. given a procedure for deciding $\leqslant$ (resp. $=$) on level i types we can construct a procedure for deciding $\leqslant$ (resp. $=$) on level i actions;
2. given a procedure for deciding $\leqslant$ on level i actions we can construct a procedure for deciding $\leqslant$ on level i behaviors;
3. given a procedure for deciding $=$ on level i actions we can construct a procedure for deciding $=$ on level i behaviors;
4. given a procedure for deciding $\leqslant$ on level i behaviors we can construct a procedure for deciding $\leqslant$ on level i behavior contexts and a procedure for deciding $\leqslant$ on level i behavior rows;
5. given a procedure for deciding $=$ on level i behaviors we can construct a procedure for deciding $=$ on level i behavior contexts and a procedure for deciding $=$ on level i behavior rows;
6. given a procedure for deciding $\leqslant$ (resp. $=$) on level $i-1$ behavior contexts/rows we can construct a procedure for deciding $\leqslant$ (resp. $=$) on level i types.

Except for 2, 3, and 4, to be treated below, these constructions are all obvious.

The case 2. Here the procedure works as follows: given b_1 and b_2 of level i, implemented by ϵ-free automata A_1 and A_2, use our assumptions and Lemma 6.17 to construct $A_1 \setminus^{\leq} A_2$. We shall prove that deciding $b_1 \leqslant b_2$ amounts to deciding whether $\mathsf{Acc}(A_1 \setminus^{\leq} A_2) = \emptyset$, a property which is decidable (Lemma 6.8).

So first assume that $b_1 \leqslant b_2$ is false. Thus there exists $tr \in b_1 = \mathsf{Acc}(A_1)$ such that for no tr^+ with $tr \leqslant tr^+$ it holds that $tr^+ \in b_2 = \mathsf{Acc}(A_2)$. By Lemma 6.18, this shows that $tr \in \mathsf{Acc}(A_1 \setminus^{\leq} A_2)$.

Conversely, assume that $\mathsf{Acc}(A_1 \setminus A_2) \neq \emptyset$. Let $tr \in \mathsf{Acc}(A_1 \setminus^{\leq} A_2)$. By Lemma 6.18, $tr \in \mathsf{Acc}(A_1) = b_1$, and for no tr^+ with $tr^+ \geqslant tr$ it holds that $tr^+ \in \mathsf{Acc}(A_2) = b_2$. Hence, $b_1 \leqslant b_2$ is false.

The case 3. Here the procedure works as follows: given b_1 and b_2 of level i, implemented by ϵ-free automata A_1 and A_2, use our assumptions and Lemma 6.17 to construct $A = A_1 \setminus A_2$ and $A' = A_2 \setminus A_1$. Now $b_1 = b_2$ iff $\mathsf{Acc}(A_1) \setminus \mathsf{Acc}(A_2) = \emptyset$ and $\mathsf{Acc}(A_2) \setminus \mathsf{Acc}(A_1) = \emptyset$ iff (by Lemma 6.18) $\mathsf{Acc}(A) = \emptyset$ and $\mathsf{Acc}(A') = \emptyset$, a property which is decidable (by Lemma 6.8).

The case 4. Here the procedure works as follows: given B_1 and B_2 of level i, use Lemma 3.12 to construct an action test of level zero with the property that $B_1 \leqslant B_2$ iff $B_1 \lfloor \{\mathsf{test}\} \rfloor \leqslant B_2 \lfloor \{\mathsf{test}\} \rfloor$, a relation which by our assumptions is decidable (since $B_1 \lfloor \{\mathsf{test}\} \rfloor$ and $B_2 \lfloor \{\mathsf{test}\} \rfloor$ are of level i). □

The decision procedures are thus potentially very expensive for entities of high level, but this in itself is no reason for discomfiture. Such entities are probably rare in practice; for instance, in the communication-free ambient calculus, types are of level 1 and all other entities are of nesting 0. Another concern is that the size of the automaton for a behavior is exponential in the number of

parallel constructs. In practice, however, there may never be more than a few processes running in parallel. In general, hope for efficiency in practice is supported by other similar situations. For example, ML type-inference shows an extreme disparity between worst-case performance in theory (exponential time) and actual performance in practice (very fast).

Corollary 6.20. *Given a behavior b, it is decidable whether $b \mid b \leqslant b$.* □

6.3 Annotated Automata

In order to check the trace correctness predicate we shall employ the notion of an *annotated automaton*, that is an automaton $(\mathcal{Q}, \iota, F, \delta)$ equipped with a function H mapping each $q \in \mathcal{Q}$ into an upwards closed set of groups $\mathsf{H}(q)$. The idea (cf. Lemmas 6.23 and 6.24) is that if q is reachable from ι via some trace tr then $\mathsf{H}(q)$ must be an upper approximation of $Dest(H_0, tr)$ where H_0 is the "initial surrounding". Formally, we have the following definition, where a *positional action* is an action of the form $\text{-}\mathsf{enter}(\_)$ or $\text{-}\mathsf{exit}(\_)$ or $\_\mathsf{open}(\_)$.

Definition 6.21. Let $A = (\mathcal{Q}, \iota, F, \delta)$ be an automaton, and H_0 an upwards closed set of groups. We say that H is a valid annotation of A wrt. H_0 iff

- $H_0 \subseteq \mathsf{H}(\iota)$; and
- $(q_1, \epsilon, q_2) \in \delta$ implies $\mathsf{H}(q_1) \subseteq \mathsf{H}(q_2)$; and
- $(q_1, a, q_2) \in \delta$, with a a positional action, implies $Dest(\mathsf{H}(q_1), a) \subseteq \mathsf{H}(q_2)$; and
- $(q_1, \mathsf{put}(\_)\,\mathsf{get}(\_), q_2) \in \delta^*$ implies $\mathsf{H}(q_1) \subseteq \mathsf{H}(q_2)$. □

Lemma 6.22 (Least valid annotation)**.** *Consider the algorithm in Fig. 8, which given A and H_0 computes an annotation $\mathsf{H}^A_{H_0}$.*

- *$\mathsf{H}^A_{H_0}$ is valid wrt. H_0; and*
- *$\mathsf{H}^A_{H_0}$ is the least annotation that is valid wrt. H_0.* □

Proof. Omitted, as it follows standard techniques for proving results about an iterative algorithm employing a "work list" (W). Note that we make use of the fact that if H_1 and H_2 are upwards closed then also $H_1 \cup H_2$ is upwards closed. □

Observe that if A is normalized then $\mathsf{H}^A_{H_0}(\iota) = H_0$.

The next two lemmas, both proved in Appendix A, formalizes the relationship between $\mathsf{H}^A_{H_0}$ and feasible traces:

Lemma 6.23. *Let $A = (\mathcal{Q}, \iota, F, \delta)$ and H_0 be given, and assume that tr is saturated. If $q \in \mathcal{Q}$ is such that $(\iota, tr, q) \in \delta^*$, then $Dest(H_0, tr) \subseteq \mathsf{H}^A_{H_0}(q)$.*

In particular, if tr is feasible from H_0 then $\mathsf{H}^A_{H_0}(q) \neq \emptyset$. □

Lemma 6.24. *Let $A = (\mathcal{Q}, \iota, F, \delta)$ and H_0 be given. If $g \in \mathsf{H}^A_{H_0}(q)$ with $q \in \mathcal{Q}$, then there exists tr feasible from H_0 with $(\iota, tr, q) \in \delta^*$ and $g \in Dest(H_0, tr)$.* □

Input: $A = (\mathcal{Q}, \iota, F, \delta)$
H_0

Output: $\mathsf{H}^A_{H_0}$

Method:

$\mathsf{H}(\iota) := H_0$
$\mathsf{H}(q) := \emptyset$ for $q \neq \iota$
$W := \{\iota\}$
while $W \neq \emptyset$ do
  select $q \in W$
  $W := W \setminus \{q\}$
  for all q' with $(q, \epsilon, q') \in \delta$ or with $(q, \mathsf{put}(\_)\,\mathsf{get}(\_), q') \in \delta^*$:
    if $\mathsf{H}(q) \subseteq \mathsf{H}(q')$
      then skip
      else $\mathsf{H}(q') ::= \mathsf{H}(q) \cup \mathsf{H}(q')$
        $W := W \cup \{q'\}$
  for all q' with $(q, a, q') \in \delta$ where a is positional:
    if $Dest(\mathsf{H}(q), a) \subseteq \mathsf{H}(q')$
      then skip
      else $\mathsf{H}(q') ::= Dest(\mathsf{H}(q), a) \cup \mathsf{H}(q')$
        $W := W \cup \{q'\}$
endwhile
$\mathsf{H}^A_{H_0} := \mathsf{H}$

Figure 8. Computing the Least Valid Annotation.

6.4 Decidability of the Trace Correctness Predicate

For deciding the trace correctness predicate we use the algorithm in Fig. 9.

Lemma 6.25. *Given input b, H_0, g, H, and $br = \{g_i : b_i\}_{i \in I}$, the algorithm in Fig. 9 always terminates, returning either "true" or "false". It returns "true" if and only if $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ holds.* □

Proof. Only the last claim is non-trivial; below we shall prove first "if" and then "only if". We shall refer to the entities of the algorithm by the names used in Fig. 9.

The algorithm returns "false" implies that $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ does not hold. We now examine each of the four places in the algorithm where the value "false" is returned.

there exists $q \in Q'$ such that $\mathsf{H}^A_{H_0}(q) \subseteq H$ does not hold. Thus there exists $g' \in \mathsf{H}^A_{H_0}(q)$ with $g' \notin H$. By Lemma 6.24, there exists tr_1 feasible from H_0 with $(\iota, tr_1, q) \in \delta^*$ and $g' \in Dest(H_0, tr_1)$. Since A is junk-free, there exists $f \in F$ and tr_2 with $(q, tr_2, f) \in \delta^*$, implying $(\iota, tr_1 \diamond tr_2, f) \in \delta^*$. Thus $tr_1 \diamond tr_2 \in \mathsf{Acc}(A) = b$, but it does not hold (as witnessed by g') that $Dest(H_0, tr_1) \subseteq H$. Therefore clause 1 in Def 3.19 is not satisfied.

there exists $q \in Q'$ and $(q, {}_G\mathsf{open}(g'), q') \in \delta$ but $g \notin G$. By Lemma 6.24, applicable since $\mathsf{H}^A_{H_0}(q) \neq \emptyset$, there exists tr_1 feasible from H_0 with $(\iota, tr_1, q) \in \delta^*$. Since A is junk-free, there exists $f \in F$ and tr_2 with $(q', tr_2, f) \in \delta^*$. Thus $tr_1 \diamond {}_G\mathsf{open}(g') \diamond tr_2 \in \mathsf{Acc}(A) = b$, but $g \notin G$. Therefore clause 4 in Def 3.19 is not satisfied.

there exists $q \in Q'$ and $(q, \mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2), q') \in \delta^*$ but $\sigma_1 \leqslant \sigma_2$ does not hold.. By Lemma 6.24, applicable since $\mathsf{H}^A_{H_0}(q) \neq \emptyset$, there exists tr_1 feasible from H_0 with $(\iota, tr_1, q) \in \delta^*$, and since A is

Input: b, H_0, g, H, $br = \{g_i : b_i\}_{i \in I}$

Output: the truth value of $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$

Method:

(Lemmas 6.5, 6.6) Construct $A = (\mathcal{Q}, \iota, F, \delta)$ implementing b, with A ϵ-free and junk-free
(Fig. 8) Compute $\mathsf{H}^A_{H_0}$
Let $Q' = \{q \in \mathcal{Q} \mid \mathsf{H}^A_{H_0}(q) \neq \emptyset\}$
for all $q \in Q'$:
if $\mathsf{H}^A_{H_0}(q) \subseteq H$
then skip
else return false
for all $(q, G\ \mathsf{open}(\_), \_) \in \delta$:
if $g \in G$
then skip
else return false
for all $(q, \mathsf{put}(\sigma_1)\ \mathsf{get}(\sigma_2), \_) \in \delta^*$:
(Theorem 6.19) if $\sigma_1 \leqslant \sigma_2$
then skip
else return false
for all $i \in I$:
(Lemma 6.5) Construct A_i implementing b_i with A_i ϵ-free
Let ι_i be a new symbol
Construct $A''_i = (\mathcal{Q} \cup \{\iota_i\}, \iota_i, F, \delta \cup \{(\iota_i, \epsilon, q) \mid g_i \in \mathsf{H}^A_{H_0}(q)\})$
(Lemma 6.5) Construct A'_i with $\mathsf{Acc}(A'_i) = \mathsf{Acc}(A''_i)$ and A'_i ϵ-free
(Def. 6.16) Compute $A'_i \setminus^{\leq} A_i$
(Lemma 6.8) if $\mathsf{Acc}(A'_i \setminus^{\leq} A_i) = \emptyset$
then skip
else return false
return true

Figure 9. Checking the Trace Correctness Predicate.

junk-free there also exists $f \in F$ and tr_2 with $(q', tr_2, f) \in \delta^*$. Thus $tr_1 \diamond \mathsf{put}(\sigma_1)\ \mathsf{get}(\sigma_2) \diamond tr_2 \in \mathsf{Acc}(A) = b$, but $\sigma_1 \leqslant \sigma_2$ does not hold. Therefore clause 3 in Def 3.19 is not satisfied.

there exists $i \in I$ **with** $\mathsf{Acc}(A'_i \setminus^{\leq} A_i) \neq \emptyset$**.** Let tr belong to $\mathsf{Acc}(A'_i \setminus^{\leq} A_i)$. By Lemma 6.18, $tr \in \mathsf{Acc}(A'_i)$ and

$$\forall tr^+ \text{ with } tr \leqslant tr^+\text{: } tr^+ \notin \mathsf{Acc}(A_i) = b_i. \tag{1}$$

Since $tr \in \mathsf{Acc}(A'_i) = \mathsf{Acc}(A''_i)$, it is easy to see that there exists $q \in Q'$ with $g_i \in \mathsf{H}^A_{H_0}(q)$ and $f \in F$ such that $(q, tr, f) \in \delta^*$. By Lemma 6.24, there exists tr_1 feasible from H_0 with $(\iota, tr_1, q) \in \delta$ and $g_i \in Dest(H_0, tr_1)$. Thus $(\iota, tr_1 \diamond tr, f) \in \delta^*$, implying $tr_1 \diamond tr \in \mathsf{Acc}(A) = b$, but (1) shows that $\{tr\} \leqslant b_i$ does not hold. Therefore clause 2 in Def 3.19 is not satisfied.

not $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ implies that the algorithm returns "false". Our assumption entails that there exists tr_1 feasible from H_0 and tr_2 such that $tr_1 \diamond tr_2 \in b = \mathsf{Acc}(A)$; we can thus find q and $f \in F$ such that $(\iota, tr_1, q) \in \delta^*$ and $(q, tr_2, f) \in \delta^*$. Let $H_1 = Dest(H_0, tr_1)$; by Lemma 6.23 we have $H_1 \subseteq \mathsf{H}^A_{H_0}(q) \neq \emptyset$, implying $q \in Q'$. Examining Def. 3.19, we see that there are four possible reasons why $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ fails:

$H_1 \subseteq H$ **does not hold.** But then neither $\mathsf{H}^A_{H_0}(q) \subseteq H$ holds, causing the algorithm to return "false".

tr_2 **takes the form** ${}_G\mathsf{open}(\_) \diamond tr_3$ **with** $g \notin G$**.** Since A is ϵ-free, there exists q' such that $(q, {}_G\,\mathsf{open}(\_), q') \in \delta$. Therefore the algorithm returns "false".

tr_2 **takes the form** $\mathsf{put}(\sigma_1)\mathsf{get}(\sigma_2)\diamond tr_3$ **with** $\sigma_1 \not\leqslant \sigma_2$**.** There exists q' with $(q, \mathsf{put}(\sigma_1)\mathsf{get}(\sigma_2), q') \in \delta^*$. Therefore the algorithm returns "false".

there exists $i \in I$ **with** $g_i \in H_1$ **but** $\{tr_2\} \leqslant b_i$ **does not hold.** Since $H_1 \subseteq \mathsf{H}^A_{H_0}(q)$ we infer that $g_i \in \mathsf{H}^A_{H_0}(q)$, which since $(q, tr_2, f) \in \delta^*$ entails that $tr_2 \in \mathsf{Acc}(A''_i) = \mathsf{Acc}(A'_i)$. By our assumption, there exists no $tr_2^+ \in b_i = \mathsf{Acc}(A_i)$ with $tr_2 \leqslant tr_2^+$, so Lemma 6.18 tells us that $tr_2 \in \mathsf{Acc}(A'_i \setminus^{\leq} A_i)$. Therefore the algorithm returns "false". □

6.5 Type checking

We are now ready for the main result of this section.

Theorem 6.26 (Decidability of type checking)**.** *Given a purported derivation of* $E \vdash M : \tau$ *or* $\Delta, E \vdash_g P : b$*, we can effectively check its validity according to the rules in Fig. 7.* □

Proof. To check that two behaviors or two types are in fact identical, and to check the side conditions for the rules (Exp Subsumption) and (Proc Subsumption), we employ Theorem 6.19. The side condition of (Exp Open) is trivially decidable. For the side condition of (Proc Repl), employ Corollary 6.20. For the essential side conditon of (Proc Amb), employ Lemma 6.25. □

It is clear, however, that the worst-case complexity of type checking could be very high.

7 Type Inference

We now briefly sketch how one might perform type *inference*, where ideally the user, in addition to the security constraints, provides only the group of each ν-bound ambient name. It seems that we need to impose several restrictions on the form of the input, none of which appears to exclude most commonly met processes:

1. only *ambients* can be replicated (as given $b_0 \neq \varepsilon$, it is not obvious how to find b with $b_0 \leqslant b$ such that $b \mid b \leqslant b$);
2. for each group g, there exists at most one g_0 such that $\mathcal{O}(g, g_0)$ (to help dealing with the non-determinism in (Exp Open));
3. no ambient names are communicated, and if capabilities are communicated then these can contain only $\mathsf{enter}()$ and $\mathsf{exit}()$ actions and the user has to provide their full type;
4. if a capability $\mathsf{open}\ n$ occurs enclosed directly within an ambient m of group g, then $\mathcal{O}(g_n, g)$ must hold where g_n is the group of n (this might not be the case if this capability is supposed to be executed *after* m itself has been opened);
5. the graph induced by the dynamic security constraints contains no cycles.

Then a type reconstruction algorithm could proceed as follows:

1. For each ν-bound ambient n with group g, define $H_g = \{g' \mid \mathcal{I}(g, g')\}$ and assign to n the (preliminary) type $\mathsf{amb}^g_{H_g}$. (Alternatively, to gain greater precision, one could proceed in an iterative way: initiate H_g to $\emptyset$, and if step 2 fails then increment H_g appropriately and start all over again.)

2. Make a topological sort on the groups (possible, thanks to restriction 5), such that if g is less than or equal to g' then $\mathcal{O}(g', g)$ does not hold. Starting with the smallest g, we find a behavior b such that if a process P is enclosed within an ambient of group g then P has behavior b (occurrences of $\mathsf{open}\ \_$ in P will not cause trouble, thanks to restriction 4). With g_0 the group (cf. restriction 2) such that $\mathcal{O}(g, g_0)$, we now find b_0 (using techniques similar to the ones used for establishing Lemma 6.25) such that $(\_, b) \overset{g}{\rightsquigarrow} (H_g, \{g_0 : b_0\})$ (this might fail, for instance if H_g is too small). Now globally update the type of all ambients of group g into $\mathsf{amb}^g_{H_g}[g_0 : b_0]$.

One can optimize the procedure sketched above in various ways, and alleviate certain of the restrictions. For example, it would be useful to allow for $\mathcal{O}(g, g)$ (cf. Gonthier's coalescing encoding of channels mentioned in [CG99]) which will give rise to equations of the form $(\_, b) \overset{g}{\rightsquigarrow} (H_g, \{g_0 : \beta_0\})$ where β_0 occurs in b. In certain cases, these can be solved (by creating a "backwards loop" in the automaton).

References

[AKPG01] T. Amtoft, A. J. Kfoury, and S. M. Pericas-Geertsen. What are polymorphically-typed ambients? In D. Sands, ed., *ESOP 2001, Genova*, vol. 2028 of *LNCS*, pp. 206–220. Springer-Verlag, Apr. 2001. An extended version appears as Technical Report BUCS-TR-2000-021, Comp.Sci. Department, Boston University, 2000. 2, 5, 6, 8, 11

[BC01] M. Bugliesi and G. Castagna. Secure safe ambients. In *Conf. Rec. POPL '01: 28th ACM Symp. Princ. of Prog. Langs.*, pp. 222–235, 2001. 2, 3, 4

[BCC01] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *CONCUR 2001*, vol. 2154 of *LNCS*, pp. 102–120. Springer-Verlag, 2001. 2

[BPG00] M. Bugliesi and S. M. Pericas-Geertsen. Depth subtyping and type inference for object calculi. In *Proc. Seventh Workshop on Foundations of Object-Oriented Languages*, Boston, Mass., U.S.A., 2000. 11

[Car99] L. Cardelli. Abstractions for mobile computation. In J. Vitek and C. Jensen, eds., *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, vol. 1603 of *LNCS*, pp. 51–94. Springer-Verlag, 1999. 4

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, ed., *FoSSaCS'98*, vol. 1378 of *LNCS*, pp. 140–155. Springer-Verlag, 1998. 1, 4

[CG99] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *POPL'99, San Antonio, Texas*, pp. 79–92. ACM Press, Jan. 1999. 1, 2, 4, 6, 34, 39

[CGG99] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., *ICALP'99*, vol. 1644 of *LNCS*, pp. 230–239. Springer-Verlag, July 1999. Extended version appears as Microsoft Research Technical Report MSR-TR-99-32, 1999. 1, 2

[CGG00] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000), Tohoku University, Sendai, Japan*, vol. 1872 of *LNCS*, pp. 333–347. Springer-Verlag, Aug. 2000. 2, 4

[DCS00] M. Dezani-Ciancaglini and I. Salvo. Security types for mobile safe ambients. In *ASIAN Computing Science Conference - ASIAN'00*, vol. 1961 of *LNCS*, pp. 215–236. Springer, 2000. 2

[GH99] S. Gay and M. Hole. Types and subtypes for client-server interactions. In *Proc. European Symp. on Programming*, vol. 1576 of *LNCS*, pp. 74–90. Springer-Verlag, 1999. 2

[GYY01] X. Guan, Y. Yang, and J. You. Typing evolving ambients. *Information Processing Letters*, 80(5):265–270, Nov. 2001. 2, 8

[HJNN99] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Abstract interpretation of mobile ambients. In *SAS'99*, vol. 1694 of *LNCS*, pp. 134–148. Springer-Verlag, 1999. 1

[LM01] F. Levi and S. Maffeis. An abstract interpretation framework for analysing mobile ambients. In *SAS'01*, vol. 2126 of *LNCS*, pp. 395–411. Springer-Verlag, 2001. 1, 2

[LS00] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL'00, Boston, Massachusetts*, pp. 352–364. ACM Press, Jan. 2000. 2

[NN94] H. R. Nielson and F. Nielson. Higher-order concurrent programs with finite communication topology. In *Conf. Rec. 21st Ann. ACM Symp. Princ. of Prog. Langs.*, pp. 84–97, 1994. 1

[NN01] H. R. Nielson and F. Nielson. Shape analysis for mobile ambients. *Nordic Journal of Computing*, 8:233–275, 2001. A preliminary version appeared at POPL'00. 1, 2

[NNHJ99] F. Nielson, H. R. Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proc. CONCUR'99*, vol. 1664 of *LNCS*, pp. 463–477. Springer-Verlag, 1999. 1, 4, 5

[NNS00] F. Nielson, H. R. Nielson, and M. Sagiv. A Kleene analysis of mobile ambients. In *Programming Languages & Systems, 9th European Symp. Programming*, vol. 1782 of *LNCS*, pp. 305–319. Springer-Verlag, 2000. 1, 2, 6

[Zim00] P. Zimmer. Subtyping and typing algorithms for mobile ambients. In *FOSSACS 2000, Berlin*, vol. 1784 of *LNCS*, pp. 375–390. Springer-Verlag, 2000. 11

A Proofs of Results in Main Text

Lemma 3.2 Given the behavior contexts B_1, B_2, and the behavior b. Then $B_1\lfloor B_2\lfloor b\rfloor\rfloor = (B_1\lfloor B_2\rfloor)\lfloor b\rfloor$.

Proof. Structural induction in B_1. If $B_1 = \Box$ the claim is trivial, as both sides evaluate to $B_2\lfloor b\rfloor$. If $B_1 = a.B_0$, the induction hypothesis enables us to derive the desired relation

$$B_1\lfloor B_2\lfloor b\rfloor\rfloor = a.(B_0\lfloor B_2\lfloor b\rfloor\rfloor) = a.((B_0\lfloor B_2\rfloor)\lfloor b\rfloor) = (a.(B_0\lfloor B_2\rfloor))\lfloor b\rfloor = (B_1\lfloor B_2\rfloor)\lfloor b\rfloor.$$

If $B_1 = b_0 \mid B_0$, the induction hypothesis enables us to derive the desired relation

$$B_1\lfloor B_2\lfloor b\rfloor\rfloor = b_0 \mid (B_0\lfloor B_2\lfloor b\rfloor\rfloor) = b_0 \mid ((B_0\lfloor B_2\rfloor)\lfloor b\rfloor) = (b_0 \mid B_0\lfloor B_2\rfloor)\lfloor b\rfloor = (B_1\lfloor B_2\rfloor)\lfloor b\rfloor.$$

□

Lemma 3.11 Given behavior context B, and an action test such that test tests B. For all behaviors b and traces tr, $tr \in B\lfloor b\rfloor$ holds if and only if there exists tr_1, tr_2, tr_0 such that $tr_1 \diamond \mathsf{test} \diamond tr_2 \in B\lfloor\{\mathsf{test}\}\rfloor$ and $tr_0 \in \{tr_2\} \mid b$ and $tr = tr_1 \diamond tr_0$.

Proof. The proof is by induction in B, where we perform a case analysis on the form of B.

$\underline{B = \square.}$ For "if", we from $tr_1 \diamond \mathsf{test} \diamond tr_2 \in \{\mathsf{test}\}$ infer that $tr_1 = tr_2 = \bullet$ and therefore $tr = tr_0 \in b$, as desired.

For "only if", we can pick $tr_1 = tr_2 = \bullet$ and $tr_0 = tr$.

$\underline{B = a.B'.}$ For "if", we infer from $tr_1 \diamond \mathsf{test} \diamond tr_2 \in a.B'\lfloor\{\mathsf{test}\}\rfloor$ that (since $a \neq \mathsf{test}$) we can write $tr_1 = a \diamond tr_1'$ with $tr_1' \diamond \mathsf{test} \diamond tr_2 \in B'\lfloor\{\mathsf{test}\}\rfloor$. We can therefore write $tr = a \diamond tr'$ with $tr' = tr_1' \diamond tr_0$. The induction hypothesis therefore tells us that $tr' \in B'\lfloor b\rfloor$, showing that $tr \in B\lfloor b\rfloor$ as desired.

For "only if", we see that we can write $tr = a \diamond tr'$ with $tr' \in B'\lfloor b\rfloor$. By applying the induction hypothesis, we find $tr_1' \diamond \mathsf{test} \diamond tr_2 \in B'\lfloor\{\mathsf{test}\}\rfloor$ and $tr_0 \in \{tr_2\} \mid b$ such that $tr' = tr_1' \diamond tr_0$. With $tr_1 = a \diamond tr_1'$, we therefore get $tr = tr_1 \diamond tr_0$ and $tr_1 \diamond \mathsf{test} \diamond tr_2 \in B\lfloor\{\mathsf{test}\}\rfloor$, as desired.

$\underline{B = b' \mid B'.}$ For "if", we infer from $tr_1 \diamond \mathsf{test} \diamond tr_2 \in b' \mid B'\lfloor\{\mathsf{test}\}\rfloor$ that (since test does not occur in b') there exists tr_1', tr_2', tr_1'', tr_2'' such that $tr_1 \in tr_1' \parallel tr_1''$ and $tr_2 \in tr_2' \parallel tr_2''$ and $tr_1' \diamond tr_2' \in b'$ and $tr_1'' \diamond \mathsf{test} \diamond tr_2''$ belongs to $B'\lfloor\mathsf{test}\rfloor$. By associativity of interleaving, we infer from $tr_0 \in \{tr_2\} \mid b$ that there exists $tr_0' \in \{tr_2''\} \mid b$ such that $tr_0 \in tr_2' \parallel tr_0'$. Using the induction hypothesis on B', we infer that $tr_1'' \diamond tr_0' \in B'\lfloor b\rfloor$. But then clearly $tr_1 \diamond tr_0 \in b' \mid B'\lfloor b\rfloor = B\lfloor b\rfloor$, as desired.

For "only if", our assumptions entail that there exists $tr' \in b'$ and $tr'' \in B'\lfloor b\rfloor$ such that $tr \in tr' \parallel tr''$. By applying the induction hypothesis, we find $tr_1'' \diamond \mathsf{test} \diamond tr_2'' \in B'\lfloor\mathsf{test}\rfloor$ such that $tr'' = tr_1'' \diamond tr_0''$ with $tr_0'' \in \{tr_2''\} \mid b$. Since $tr \in tr' \parallel (tr_1'' \diamond tr_0'')$ there clearly exists tr_1', tr_1, tr_0', tr_0 such that $tr' = tr_1' \diamond tr_0'$, $tr = tr_1 \diamond tr_0$, $tr_1 \in tr_1' \parallel tr_1''$, $tr_0 \in tr_0' \parallel tr_0''$. From associativity of interleaving, we infer that there exists $tr_2 \in tr_0' \parallel tr_2''$ such that $tr_0 \in \{tr_2\} \mid b$. This establishes the desired relation $tr_1 \diamond \mathsf{test} \diamond tr_2 \in (tr_1' \diamond tr_0') \parallel (tr_1'' \diamond \mathsf{test} \diamond tr_2'') \subseteq b' \mid B'\lfloor test\rfloor = B\lfloor\mathsf{test}\rfloor$. □

Lemma 3.23 If $(H_0, b) \overset{g}{\rightsquigarrow} (H, \{g_i : b_i\}_{i\in I})$ then $H_0 \subseteq H$, and for all $i \in I$ with $g_i \in H_0$ we have $b \leqslant b_i$.

Proof. If $H_0 = \emptyset$, the claim is trivial. Otherwise, $\bullet$ is feasible from H_0, with $Dest(H_0, \bullet) = H_0$. Let tr_2 be a trace in b (by our non-emptiness condition, such a trace exists). Thus the conditions mentioned in Definition 3.19 are satisfied (with $\bullet$ playing the role of tr_1), and we infer that $H_0 \subseteq H$ and that $g_i \in H_0$ implies $\{tr_2\} \leqslant b_i$. As tr_2 was an arbitrary member of b, this shows $b \leqslant b_i$ as desired. □

Lemma 3.24 The predicate $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ is anti-monotonic in H_0 and b, and monotonic in H and br.

Proof. Assume that $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$. Let $H_0^- \subseteq H_0$, $b^- \leqslant b$, $H \subseteq H^+$, and $br \leqslant br^+$; our task is to show $(H_0^-, b^-) \overset{g}{\rightsquigarrow} (H^+, br^+)$.

So let $tr_1^- \diamond tr_2^- \in b^-$ with tr_1^- feasible from H_0^-, and let $H_1^- = Dest(H_0^-, tr_1^-)$.

It is easy to see that there exists $tr_1 \diamond tr_2 \in b$ with $tr_1^- \leqslant tr_1$ and $tr_2^- \leqslant tr_2$. And with $H_1 = Dest(H_0, tr_1)$, we have $H_1^- \subseteq H_1$ (by Lemma 3.13). We infer that tr_1 is feasible from H_0.

With $br = \{g_i : b_i\}_{i \in I}$, our assumptions thus entail that

1. $H_1 \subseteq H$, and
2. for all $i \in I$: $g_i \in H_1$ implies $\{tr_2\} \leqslant b_i$, and
3. if $tr_2 = \mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2) \diamond tr_3$ then $\sigma_1 \leqslant \sigma_2$, and
4. if $tr_2 = {}_G\mathsf{open}(\_) \diamond tr_3$ then $g \in G$.

With $br^+ = \{g'_j : b'_j\}_{j \in J}$, the claim now follows from the following observations:

1. $H_1^- \subseteq H_1 \subseteq H \subseteq H^+$.
2. If $j \in J$ is such that $g'_j \in H_1^-$, then by the definition of the ordering relation on behavior rows there exists $i \in I$ with $g_i = g'_j$ and $b_i \leqslant b'_j$. Since $g_i \in H_1^- \subseteq H_1$, we from (2) above infer that $\{tr_2\} \leqslant b_i$, and therefore $\{tr_2^-\} \leqslant \{tr_2\} \leqslant b_i \leqslant b'_j$.
3. If tr_2^- takes the form $\mathsf{put}(\sigma_1^-)\,\mathsf{get}(\sigma_2^+) \diamond tr_3^-$, then tr_2 takes the form $\mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2) \diamond tr_3$ with $\sigma_1^- \leqslant \sigma_1$ and $\sigma_2 \leqslant \sigma_2^+$ (and $tr_3^- \leqslant tr_3$). Using (3) above we infer that $\sigma_1^- \leqslant \sigma_1 \leqslant \sigma_2 \leqslant \sigma_2^+$.
4. if tr_2^- takes the form ${}_{G^+}\mathsf{open}(\_) \diamond tr_3^-$, then tr_2 takes the form ${}_G\mathsf{open}(\_) \diamond tr_3$ with $G \subseteq G^+$ (and $tr_3^- \leqslant tr_3$). Using (4) above we thus infer that $g \in G$ and therefore $g \in G^+$.

(Note that the polarities of $\mathsf{put}(\_)$, $\mathsf{get}(\_)$, and $\_\mathsf{open}(\_)$ have been put to good use.) □

Lemma 3.25 Let tr be of the form either $^-\mathsf{enter}(\_)$, $^-\mathsf{exit}(\_)$, $\_\mathsf{open}(\_)$, or $\mathsf{put}(\_)\,\mathsf{get}(\_)$. Assume that

$$(H_0, tr.b) \overset{g}{\rightsquigarrow} (H, br)$$

Then with $H_1 = Dest(H_0, tr)$ we also have

$$(H_1, b) \overset{g}{\rightsquigarrow} (H, br).$$

Proof. Let $tr_1 \diamond tr_2$ belong to b, with tr_1 feasible from H_1, and let $H_2 = Dest(H_1, tr_1)$. As $H_2 = Dest(H_0, tr \diamond tr_1)$ (by Lemma 3.14) we (also employing the conditions on the form of tr) infer that $tr \diamond tr_1$ is feasible from H_0. Since $tr \diamond tr_1 \diamond tr_2 \in tr.b$, the premise of the lemma thus assures that

1. $H_2 \leqslant H$, and
2. for all $i \in I$: $g_i \in H_2$ implies $\{tr_2\} \leqslant b_i$, where $br = \{g_i : b_i\}_{i \in I}$, and
3. if tr_2 takes the form $\mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2) \diamond tr_3$ then $\sigma_1 \leqslant \sigma_2$, and
4. if tr_2 takes the form ${}_G\mathsf{open}(\_) \diamond tr_3$ then $g \in G$

But this is exactly what is needed to establish the conclusion of the lemma. □

Lemma 5.2 Assume that $E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash M : \tau$ (with $n_1 \ldots n_k$ distinct), and that there exists $V_1 \ldots V_k$ such that for all $i \in \{1 \ldots k\}$ it holds that $E \vdash V_i : \tau_i$ and that M is non-conflicting with $\{n_i\} \cup \mathsf{names}(V_i)$. Then $E \vdash M[n_i := V_i] : \tau$. Similarly, if $\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_g P : b$ and for all i it holds that $E \vdash V_i : \tau_i$ and that P is non-conflicting with $\{n_i\} \cup \mathsf{names}(V_i)$ then $\Delta, E \vdash_g P[n_i := V_i] : b$.

Proof. The proof is by induction in the size of the typing derivation for M (resp. P). We do a case analysis on the inference rule applied, but only list two of the cases; the remaining follow by straightforward applications of the induction hypothesis, except for (Proc Input) which is handled as (Proc Res) below.

(Exp n). Here M is a name n. If $n = n_i$ for some $i \in \{1 \dots k\}$, we infer that $\tau = \tau_i$. The claim is then $E \vdash V_i : \tau_i$, which is among our assumptions. If $n \neq n_i$ for all $i \in \{1 \dots k\}$, we infer that $\tau = E(n)$. But then we have $E \vdash n : \tau$, as desired.

(Proc Res). Here P takes the form $(\nu n : \tau_0).P_0$, and $E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash P : \tau$ holds because

$$E, n_1 : \tau_1, \cdots, n_k : \tau_k, n : \tau_0 \vdash P_0 : \tau. \tag{1}$$

Note that for all $i \in \{1 \dots k\}$ it holds (since P is non-conflicting with $\{n_i\} \cup \mathsf{names}(V_i)$) that $n_i \neq n$ and $n \notin \mathsf{names}(V_i)$ and that P_0 is non-conflicting with $\{n_i\} \cup \mathsf{names}(V_i)$. We can thus apply Lemma 4.5 to infer that $E, n : \tau \vdash V_i : \tau_i$, and (repeatedly) apply Lemma 4.4 to infer that

$$E, n : \tau_0, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash P_0 : \tau \tag{2}$$

by a derivation of the same shape as the one for (1). We can thus apply the induction hypothesis on (2) to infer that $E, n : \tau_0 \vdash P_0[n_i := V_i] : \tau$. But then an application of (Proc Res) yields $E \vdash (\nu n : \tau_0).P_0[n_i := V_i] : \tau$, which is as desired since $P[n_i := V_i] = (\nu n : \tau_0).P_0[n_i := V_i]$. □

Lemma 5.4 Assume that

$$\Delta, E \vdash_g \mathcal{PC}\lfloor P \rfloor_p : b.$$

Then there exists E_0, g_0, and b_0 such that

$$\Delta, E_0 \vdash_{g_0} P : b_0$$

and with the property that if there also exists a judgement

$$\Delta', E_0 \vdash_{g_0} Q : b_0$$

where $\Delta \underset{\overline{\mathcal{PC}}}{=\!=} \Delta'$, then also

$$\Delta', E \vdash_g \mathcal{PC}\lfloor Q \rfloor_p : b.$$

Proof. The proof is by induction in $\mathcal{PC}$. We do a case analysis on $\mathcal{PC}$, where only four cases are possible (as $\mathcal{PC}$ expects a process, not an expression, in its hole):

$\mathcal{PC} = \Box_p$. Choose $E_0 = E$, $g_0 = g$, and $b_0 = b$. Then the claim clearly holds.

$\mathcal{PC} = \mathcal{PC}_0 \mid R$. We assume $\Delta, E \vdash_g \mathcal{PC}_0\lfloor P \rfloor_p \mid R : b$, so there exists b_1 and b_2 with $b_1 \mid b_2 \leqslant b$ such that

$$\Delta, E \vdash_g \mathcal{PC}_0\lfloor P \rfloor_p : b_1$$

and $\Delta, E \vdash_g R : b_2$. We can thus apply the induction hypothesis to find E_0, g_0, and b_0 such that $\Delta, E_0 \vdash_{g_0} P : b_0$.

Now assume that $\Delta', E_0 \vdash_{g_0} Q : b_0$ where $\Delta \underset{\overline{\mathcal{PC}}}{=\!=} \Delta'$. Thus $\Delta \underset{\overline{\mathcal{PC}_0}}{=\!=} \Delta'$ so we can thus further apply the induction hypothesis to infer

$$\Delta', E \vdash_g \mathcal{PC}_0\lfloor Q \rfloor_p : b_1$$

and also $\Delta \underset{R}{=} \Delta'$ so we can apply Lemma 4.3 to infer $\Delta', E \vdash_g R : b_2$. But this enables us to arrive at the judgement

$$\Delta', E \vdash_g \mathcal{PC}_0\lfloor Q \rfloor_p \mid R : b$$

which is as desired since $\mathcal{PC}\lfloor Q \rfloor_p = \mathcal{PC}_0\lfloor Q \rfloor_p \mid R$.

$\mathcal{PC} = (\nu n : \tau).\mathcal{PC}_0$. We assume $\Delta, E \vdash_g (\nu n : \tau).\mathcal{PC}_0\lfloor P \rfloor_p : b$, so we also have

$$\Delta, E, n : \tau \vdash_g \mathcal{PC}_0\lfloor P \rfloor_p : b.$$

Inductively there thus exists E_0, g_0, and b_0 such that $\Delta, E_0 \vdash_{g_0} P : b_0$.

Now assume that $\Delta', E_0 \vdash_{g_0} Q : b_0$ where $\Delta \underset{\mathcal{PC}}{=\!=} \Delta'$. Thus $\Delta \underset{\mathcal{PC}_0}{=\!=} \Delta'$ so a further application of the induction hypothesis tells us that

$$\Delta', E, n : \tau \vdash_g \mathcal{PC}_0\lfloor Q \rfloor_p : b$$

from which we arrive at $\Delta', E \vdash_g (\nu n : \tau).\mathcal{PC}_0\lfloor Q \rfloor_p : b$ which is as desired, since $\mathcal{PC}\lfloor Q \rfloor_p = (\nu n : \tau).\mathcal{PC}_0\lfloor Q \rfloor_p$.

$\mathcal{PC} = n[\mathcal{PC}_0]^{\xi}$. Our assumption is that $\Delta, E \vdash_g n[\mathcal{PC}_0\lfloor P \rfloor_p]^{\xi} : b$ from which we infer that $\varepsilon \leqslant b$ and that there exists judgements $E \vdash n : \mathsf{amb}_H^{g_n}[br]$ and

$$\Delta, E \vdash_{g_n} \mathcal{PC}_0\lfloor P \rfloor_p : b_n$$

where

$$\Delta(\xi) = b_n \text{ and } group(\xi) = g_n \text{ and } (g^{\uparrow}, b_n) \overset{g_n}{\rightsquigarrow} (H, br)$$

Inductively there exists E_0, g_0, and b_0 such that $\Delta, E_0 \vdash_{g_0} P : b_0$.

Now assume that $\Delta', E_0 \vdash_{g_0} Q : b_0$ where $\Delta \underset{\mathcal{PC}}{=\!=} \Delta'$. Thus $\Delta \underset{\mathcal{PC}_0}{=\!=} \Delta'$ so we can further apply the induction hypothesis to infer

$$\Delta', E \vdash_{g_n} \mathcal{PC}_0\lfloor Q \rfloor_p : b_n$$

and also $\Delta(\xi) = \Delta'(\xi)$ so we have $\Delta'(\xi) = b_n$. This enables us to arrive at

$$\Delta', E \vdash_g n[\mathcal{PC}_0\lfloor Q \rfloor_p]^{\xi} : b$$

which is as desired since $\mathcal{PC}\lfloor Q \rfloor_p = n[\mathcal{PC}_0\lfloor Q \rfloor_p]^{\xi}$. □

Lemma 5.5 Suppose that $P \equiv Q$. Then $\Delta, E \vdash_g P : b$ if and only if $\Delta, E \vdash_g Q : b$.

Proof. The proof is by induction on the derivation of $P \equiv Q$, much as the similar result in [CG99]. Below we list the most interesting cases.

(Struct ParComm). Follows from commutativity of " | ".

(Struct ParAssoc). Assume that $\Delta, E \vdash_g (P \mid Q) \mid R : b$ (the other direction is similar). Then there exists b_{12} and b_3 with $b_{12} \mid b_3 \leqslant b$ such that $\Delta, E \vdash_g P \mid Q : b_{12}$ and $\Delta, E \vdash_g R : b_3$. Therefore there exists b_1 and b_2 with $b_1 \mid b_2 \leqslant b_{12}$ such that $\Delta, E \vdash_g P : b_1$ and $\Delta, E \vdash_g Q : b_2$. Now $\Delta, E \vdash_g P \mid (Q \mid R) : b_1 \mid (b_2 \mid b_3)$, and by Lemma 3.6 we infer that $b_1 \mid (b_2 \mid b_3) \equiv (b_1 \mid b_2) \mid b_3 \leqslant b_{12} \mid b_3 \leqslant b$. This implies the desired judgment $\Delta, E \vdash_g P \mid (Q \mid R) : b$.

(Struct ResRes). Follows easily from Lemma 4.4.

(Struct ResPar). Assume that $\Delta, E \vdash_g (\nu n:\tau).(P \mid Q) : b$. There exists b_1 and b_2 with $b_1 \mid b_2 \leqslant b$ such that $\Delta, E, n:\tau \vdash_g P : b_1$ and $\Delta, E, n:\tau \vdash_g Q : b_2$. Since $n \notin \mathsf{fn}(P)$, we can α-rename P into a P' such that $n \notin \mathsf{names}(P')$. Clearly $\Delta, E, n:\tau \vdash_g P' : b_1$ (this claim amounts to the case (Struct α-rename)), so by Lemma 4.6 we have $\Delta, E \vdash_g P' : b_1$ and therefore also $\Delta, E \vdash_g P : b_1$. We can thus infer first $\Delta, E \vdash_g (\nu n:\tau).Q : b_2$ and then (since $b_1 \mid b_2 \leqslant b$) the desired judgment $\Delta, E \vdash_g P \mid (\nu n:\tau).Q : b$.

For the other direction, assume that $\Delta, E \vdash_g P \mid (\nu n:\tau).Q : b$. There exists b_1 and b_2 with $b_1 \mid b_2 \leqslant b$ such that $\Delta, E \vdash_g P : b_1$ and $\Delta, E, n:\tau \vdash_g Q : b_2$. Since $n \notin \mathsf{fn}(P)$, we can α-rename P into a P' such that $n \notin \mathsf{names}(P')$. Then $\Delta, E \vdash_g P' : b_1$ so by Lemma 4.5 we have $\Delta, E, n:\tau \vdash_g P' : b_1$ and therefore also $\Delta, E, n:\tau \vdash_g P : b_1$. We can thus infer first $\Delta, E, n:\tau \vdash_g P \mid Q : b_1 \mid b_2$ and then the desired judgement $\Delta, E \vdash_g (\nu n:\tau).(P \mid Q) : b$.

(Struct ResAmb). Let $E(m) = (E, n:\tau)(m) = \mathsf{amb}_H^{g_0}\lfloor br \rfloor$. Assume that $\Delta, E \vdash_g (\nu n:\tau).m[\![P]\!]^\xi : b$ (the other direction is similar). We infer that $\varepsilon \leqslant b$, and that there exists b_0 such that $\Delta, E, n:\tau \vdash_{g_0} P : b_0$ and $(g^\uparrow, b_0) \overset{g_0}{\rightsquigarrow} (H, br)$ with $group(\xi) = g_0$ and $\Delta(\xi) = b_0$. But then $\Delta, E \vdash_{g_0} (\nu n:\tau).P : b_0$, clearly enabling us to infer the desired judgment $\Delta, E \vdash_g m[\![(\nu n:\tau).P]\!]^\xi : b$.

(Struct ZeroPar). Assume that $\Delta, E \vdash_g P \mid \mathbf{0} : b$. There exists b_1 and b_2 with $b_1 \mid b_2 \leqslant b$ such that $\Delta, E \vdash_g P : b_1$ and $\Delta, E \vdash_g \mathbf{0} : b_2$. We infer that $\varepsilon \leqslant b_2$ and by Lemma 3.6 therefore $b_1 \equiv b_1 \mid \varepsilon \leqslant b_1 \mid b_2 \leqslant b$. This implies the desired judgment $\Delta, E \vdash_g P : b$.

The other direction is trivial.

(Struct ZeroRepl). First assume that $\Delta, E \vdash_g !\mathbf{0} : b$. We infer that there exists $b_0 \leqslant b$ with $b_0 \mid b_0 \leqslant b_0$ such that $\Delta, E \vdash_g \mathbf{0} : b_0$, from which we further infer that $\varepsilon \leqslant b_0$. But then also $\Delta, E \vdash_g \mathbf{0} : b$, as desired, since $\varepsilon \leqslant \varepsilon \mid \varepsilon \leqslant b_0 \mid b_0 \leqslant b_0 \leqslant b$.

Conversely, assume that $\Delta, E \vdash_g \mathbf{0} : b$ from which we infer that $\varepsilon \leqslant b$. Since $\varepsilon \mid \varepsilon \leqslant \varepsilon$ we can apply (Proc Zero) and (Proc Repl) to derive $\Delta, E \vdash_g !\mathbf{0} : \varepsilon$, and therefore also the desired $\Delta, E \vdash_g !\mathbf{0} : b$.

(Struct ε). Assume that $\Delta, E \vdash_g P : b$. Since $\Delta, E \vdash_g \epsilon : \mathsf{cap}[\Box]$, we can by (Proc Action) infer $\Delta, E \vdash_g \epsilon.P : b$.

Conversely, assume that $\Delta, E \vdash_g \epsilon.P : b$. Then there exists B and b_0 with $B\lfloor b_0 \rfloor \leqslant b$ such that $\Delta, E \vdash_g \epsilon : \mathsf{cap}[B]$ and $\Delta, E \vdash_g P : b_0$. As $\mathsf{cap}[\Box] \leqslant \mathsf{cap}[B]$ we deduce that $\Box \leqslant B$, implying (by Lemma 3.12) that $b_0 = \Box\lfloor b_0 \rfloor \leqslant B\lfloor b_0 \rfloor \leqslant b$. Thus we can derive the desired judgment $\Delta, E \vdash_g P : b$.

(Struct .). Assume that $\Delta, E \vdash_g (M_1.M_2).P : b$. There exists B_0 and b_0 with $B_0\lfloor b_0 \rfloor \leqslant b$ such that $E \vdash M_1.M_2 : \mathsf{cap}[B_0]$ and $\Delta, E \vdash_g P : b_0$. There thus exists B_1 and B_2 with $\mathsf{cap}[B_1\lfloor B_2 \rfloor] \leqslant \mathsf{cap}[B_0]$, implying $B_1\lfloor B_2 \rfloor \leqslant B_0$, such that $E \vdash M_1 : \mathsf{cap}[B_1]$ and $E \vdash M_2 : \mathsf{cap}[B_2]$. We can thus infer first $\Delta, E \vdash_g M_2.P : B_2\lfloor b_0 \rfloor$ and then $\Delta, E \vdash_g M_1.(M_2.P) : B_1\lfloor B_2\lfloor b_0 \rfloor \rfloor$. But by employing Lemmas 3.2 and 3.12 we deduce that $B_1\lfloor B_2\lfloor b_0 \rfloor \rfloor = (B_1\lfloor B_2 \rfloor)\lfloor b_0 \rfloor \leqslant B_0\lfloor b_0 \rfloor \leqslant b$. This shows that we can derive the desired judgment $\Delta, E \vdash_g M_1.(M_2.P) : b$.

Conversely, assume that $\Delta, E \vdash_g M_1.(M_2.P) : b$. There exists B_1 and b_1 with $B_1\lfloor b_1 \rfloor \leqslant b$ such that $E \vdash M_1 : \mathsf{cap}[B_1]$ and $\Delta, E \vdash_g M_2.P : b_1$, and therefore there exists B_2 and b_0 with $B_2\lfloor b_0 \rfloor \leqslant b_1$ such that $E \vdash M_2 : \mathsf{cap}[B_2]$ and $\Delta, E \vdash_g P : b_0$. We can thus infer first $E \vdash M_1.M_2 : \mathsf{cap}[B_1\lfloor B_2 \rfloor]$ and then $\Delta, E \vdash_g (M_1.M_2).P : (B_1\lfloor B_2 \rfloor)\lfloor b_0 \rfloor$. But by employing Lemmas 3.2 and 3.7, we deduce that $(B_1\lfloor B_2 \rfloor)\lfloor b_0 \rfloor = B_1\lfloor B_2\lfloor b_0 \rfloor \rfloor \leqslant B_1\lfloor b_1 \rfloor \leqslant b$. This shows that we can derive the desired judgment $\Delta, E \vdash_g (M_1.M_2).P : b$. □

Lemma 6.5 Given automaton A, we can construct ϵ-free A' with $\mathsf{Acc}(A') = \mathsf{Acc}(A)$.

Proof. Let $A = (\mathcal{Q}, \iota, F, \delta)$. For $q \in \mathcal{Q}$, define $\epsilon(q) = \{q' \in \mathcal{Q} \mid (q, \bullet, q') \in \delta^*\}$. Now define $A' = (\mathcal{Q}', \epsilon(\iota), F', \delta')$ where $\mathcal{Q}' = \{\epsilon(q) \mid q \in \mathcal{Q}\}$, where $Q \in F'$ iff $Q \cap F \neq \emptyset$, and where $(Q_1, a, Q_2) \in \delta'$ iff there exists $q_1 \in Q_1$ with $(q_1, a, q_2) \in \delta$ and $Q_2 = \epsilon(q_2)$.

By construction, A' is ϵ-free. To show that $\mathsf{Acc}(A') = \mathsf{Acc}(A)$, let $tr = a_1 \ldots a_n$ (with $n \geq 0$) be given. First assume that $tr \in \mathsf{Acc}(A)$. We infer that there exists $f \in F$ such that $(\iota, tr, f) \in \delta^*$, and further infer that for $i \in \{1 \ldots n\}$ there exists q_i and q_i' such that

$$(q_{i-1}, \bullet, q_i') \in \delta^* \text{ and } (q_i', a_i, q_i) \in \delta$$

where $q_0 = \iota$ and where $(q_n, \bullet, f) \in \delta^*$, implying $f \in \epsilon(q_n)$ and therefore $\epsilon(q_n) \in F'$. Therefore, for $i \in \{1 \ldots n\}$ we obtain (since $q_i' \in \epsilon(q_{i-1})$) that $(\epsilon(q_{i-1}), a_i, \epsilon(q_i)) \in \delta'$. This shows the desired $tr \in \mathsf{Acc}(A')$.

Conversely, assume that $tr \in \mathsf{Acc}(A')$. Thus, with $Q_0 = \epsilon(q_0)$ where $q_0 = \iota$ and with $Q_n \cap F \neq \emptyset$, for $i \in \{1 \ldots n\}$ there exists Q_i such that $(Q_{i-1}, a_i, Q_i) \in \delta'$, that is there exists $q_i' \in Q_{i-1}$ and q_i with $\epsilon(q_i) = Q_i$ such that $(q_i', a_i, q_i) \in \delta$. We infer that for $i \in \{1 \ldots n\}$ we have $(q_{i-1}, \bullet, q_i') \in \delta^*$, and since $\epsilon(q_n) \cap F \neq \emptyset$ there exists $f \in F$ with $(q_n, \bullet, f) \in \delta^*$. This shows that $(q_0, tr, f) \in \delta^*$, which amounts to the desired $tr \in \mathsf{Acc}(A)$. □

Lemma 6.14 $\mathsf{Acc}(A_1 \mid A_2) = \mathsf{Acc}(A_1) \mid \mathsf{Acc}(A_2)$. Moreover, $A_1 \mid A_2$ is ϵ-free, and if A_1 and A_2 are junk-free (resp. normalized) then $A_1 \mid A_2$ is junk-free (resp. normalized).

Proof. It is obvious that $A_1 \mid A_2$ is ϵ-free, and that it is normalized if A_1 and A_2 are. To reason about $A_1 \mid A_2 = (\mathcal{Q}, \iota, F, \delta)$, where for $j = 1, 2$ we have $A_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$, we first establish

$$(q_1, tr_1, q_1') \in \delta_1^* \wedge (q_2, tr_2, q_2') \in \delta_2^* \wedge tr \in tr_1 \parallel tr_2 \Longrightarrow ((q_1, q_2), tr, (q_1', q_2')) \in \delta^*. \tag{3}$$

We prove (3) by induction on the length of tr. If $tr = \bullet$, then also $tr_1 = tr_2 = \bullet$ so (exploiting that A_1 and A_2 are ϵ-free) we infer that $q_1' = q_1$ and $q_2' = q_2$ and the claim is trivial.

If tr takes the form $a \diamond tr'$, we can wlog. assume that there exists tr_1' such that $tr_1 = a \diamond tr_1'$ and $tr' \in tr_1' \parallel tr_2$. Thus (exploiting that A_1 is ϵ-free) there exists $q_1'' \in \mathcal{Q}_1$ such that $(q_1, a, q_1'') \in \delta_1$, implying $((q_1, q_2), a, (q_1'', q_2)) \in \delta$, and $(q_1'', tr_1', q_1') \in \delta_1^*$. The induction hypothesis tells us that $((q_1'', q_2), tr', (q_1', q_2')) \in \delta^*$, enabling us to arrive at the desired relation $((q_1, q_2), tr, (q_1', q_2')) \in \delta^*$.

Next we establish, also by induction on the length of tr (and exploiting that $A_1 \mid A_2$ is ϵ-free):

$$((q_1, q_2), tr, (q_1', q_2')) \in \delta^* \Longrightarrow \exists tr_1, tr_2\colon\ tr \in tr_1 \parallel tr_2 \wedge (q_1, tr_1, q_1') \in \delta_1^* \wedge (q_2, tr_2, q_2') \in \delta_2^*. \tag{4}$$

If $tr = \bullet$, then $q_1' = q_1$ and $q_2' = q_2$. We can thus choose $tr_1 = tr_2 = \bullet$.

If tr takes the form $a \diamond tr'$, then there exists q such that $((q_1, q_2), a, q) \in \delta$ and $(q, tr', (q_1', q_2')) \in \delta^*$. Wlog. we can assume that there exists q_2'' such that $q = (q_1, q_2'')$ and $(q_2, a, q_2'') \in \delta_2$. The induction hypothesis tells us that there exists tr_1 and tr_2' with $tr' \in tr_1 \parallel tr_2'$ such that $(q_1, tr_1, q_1') \in \delta_1^*$ and $(q_2'', tr_2', q_2') \in \delta_2^*$. Let $tr_2 = a \diamond tr_2'$. Then clearly $tr \in tr_1 \parallel tr_2$ and $(q_2, tr_2, q_2') \in \delta_2^*$, as desired.

We are now ready to embark on the proof that $\mathsf{Acc}(A_1 \mid A_2) = \mathsf{Acc}(A_1) \mid \mathsf{Acc}(A_2)$. If $tr \in \mathsf{Acc}(A_1 \mid A_2)$, there exists $(f_1, f_2) \in F$ such that $((\iota_1, \iota_2), tr, (f_1, f_2)) \in \delta^*$. By (4) we infer that there exists tr_1, tr_2 with $tr \in tr_1 \parallel tr_2$ such that $(\iota_1, tr_1, f_1) \in \delta_1^*$ and $(\iota_2, tr_2, f_2) \in \delta_2^*$. Since $f_1 \in F_1$ and $f_2 \in F_2$, this shows that $tr_1 \in \mathsf{Acc}(A_1)$ and that $tr_2 \in \mathsf{Acc}(A_2)$, as desired.

Conversely, if $tr \in \mathsf{Acc}(A_1) \mid \mathsf{Acc}(A_2)$ there exists $tr_1 \in \mathsf{Acc}(A_1)$ and $tr_2 \in \mathsf{Acc}(A_2)$ with $tr \in tr_1 \parallel tr_2$. Thus there exists $f_1 \in F_1$ and $f_2 \in F_2$ such that $(\iota_1, tr_1, f_1) \in \delta_1^*$ and $(\iota_2, tr_2, f_2) \in \delta_2^*$. By (3) we infer that $(\iota, tr, (f_1, f_2)) \in \delta^*$. Since $(f_1, f_2) \in F$, this shows that $tr \in \mathsf{Acc}(A)$, as desired.

We are left with showing that $A_1 \mid A_2$ is junk-free if A_1 and A_2 are; for that purpose let $q = (q_1, q_2) \in \mathcal{Q}$. By assumption, there exists tr_1,tr_2, $f_1 \in F_1$, and $f_2 \in F_2$, such that $(q_1, tr_1, f_1) \in \delta_1^*$ and $(q_2, tr_2, f_2) \in \delta_1^*$. Let $tr = tr_1 \diamond tr_2$ (or any interleaving of tr_1 and tr_2), then we infer by (3) that $(q, tr, (f_1, f_2)) \in \delta^*$. As $(f_1, f_2) \in F$, this is as desired. □

Lemma 6.18 $\mathsf{Acc}(A_1 \setminus A_2) = \mathsf{Acc}(A_1) \setminus \mathsf{Acc}(A_2)$.
$\mathsf{Acc}(A_1 \setminus^{\leq} A_2) = \{tr \in \mathsf{Acc}(A_1) \mid \mathsf{not}(\exists tr^+ \in \mathsf{Acc}(A_2) : tr \leqslant tr^+)\}$.

Proof. First we shall prove the second line. To reason about $A_1 \setminus^{\leq} A_2 = (\mathcal{Q}, \iota, F, \delta)$, where for $j = 1, 2$ we have $A_j = (\mathcal{Q}_j, \iota_j, F_j, \delta_j)$, we establish several properties, all to be proved by induction on the length of tr (and exploiting that A_1 and A_2, and hence $A_1 \setminus^{\leq} A_2$, are ϵ-free). First we establish

$$((q_1, Q_2), tr, (q_1', Q_2')) \in \delta^* \Longrightarrow (q_1, tr, q_1') \in \delta_1^*. \tag{5}$$

For if $tr = \bullet$, then $q_1 = q_1'$ and the claim follows. If $tr = a \diamond tr'$, there exists (q_1'', Q_2'') such that $((q_1, Q_2), a, (q_1'', Q_2'')) \in \delta$ and $((q_1'', Q_2''), tr', (q_1', Q_2')) \in \delta^*$, implying that $(q_1, a, q_1'') \in \delta_1$ and (inductively) that $(q_1'', tr', q_1') \in \delta_1^*$. This shows that $(q_1, tr, q_1') \in \delta_1^*$, establishing (5).

Next we establish

$$((q_1, Q_2), tr, (q_1', Q_2')) \in \delta^* \Longrightarrow ((q_2, tr^+, q_2') \in \delta_2^* \wedge tr \leqslant tr^+ \wedge q_2 \in Q_2) \Longrightarrow q_2' \in Q_2'. \tag{6}$$

For if $tr = \bullet$, we infer that $Q_2 = Q_2'$, $tr^+ = \bullet$, $q_2 = q_2'$. Thus the claim is clear.

If tr takes the form $a \diamond tr_0$, there exists q_1'' and Q_2'' such that

$$((q_1, Q_2), a, (q_1'', Q_2'')) \in \delta \text{ and} \tag{7}$$

$$((q_1'', Q_2''), tr_0, (q_1', Q_2')) \in \delta^*. \tag{8}$$

Assume that $(q_2, tr^+, q_2') \in \delta_2^*$ with $q_2 \in Q_2$ and $tr \leqslant tr^+$. We can clearly write tr^+ on the form $a^+ \diamond tr_0^+$ with $a \leqslant a^+$ and $tr_0 \leqslant tr_0^+$, and there thus exists q_2'' such that $(q_2, a^+, q_2'') \in \delta_2$ and $(q_2'', tr_0^+, q_2') \in \delta_2^*$. From this we infer using (7) that $q_2'' \in Q_2''$, and therefore by applying the induction hypothesis to (8) we infer $q_2' \in Q_2'$ thus establishing (6).

Finally, we establish

$$\begin{array}{l}(q_1, tr, q_1') \in \delta_1^* \ \wedge \ Q_2' = \{q_2' \mid \exists tr^+ \geqslant tr, q_2 \in Q_2 : (q_2, tr^+, q_2') \in \delta_2^*\} \\ \Longrightarrow ((q_1, Q_2), tr, (q_1', Q_2')) \in \delta^*.\end{array} \tag{9}$$

For if $tr = \bullet$, then $q_1' = q_1$ and it is easy to see that $Q_2' = Q_2$, yielding the claim.

Now assume that tr takes the form $a \diamond tr_0$. First define

$$Q_2'' = \{q_2'' \mid \exists a^+ \geqslant a \text{ and } q_2 \in Q_2 : (q_2, a^+, q_2'') \in \delta_2\}$$

and note that it is easy to verify that

$$Q_2' = \{q_2' \mid \exists tr_0^+ \geqslant tr_0 \text{ and } q_2'' \in Q_2'' : (q_2'', tr_0^+, q_2') \in \delta_2^*\}. \tag{10}$$

Next observe that there exists q_1'' such that $(q_1, a, q_1'') \in \delta_1$ and $(q_1'', tr_0, q_1') \in \delta_1^*$. By the construction of δ, the former relation implies that $((q_1, Q_2), a, (q_1'', Q_2'')) \in \delta$. By the induction hypothesis, the latter relation together with (10) implies that $((q_1'', Q_2''), tr_0, (q_1', Q_2')) \in \delta^*$. Therefore $((q_1, Q_2), tr, (q_1', Q_2')) \in \delta^*$, establishing (9).

We are finally ready to prove the second line of Lemma 6.18.

For "$\subseteq$", assume that $tr \in \mathsf{Acc}(A_1 \setminus^{\leq} A_2)$, that is $((\iota_1, \{\iota_2\}), tr, (f_1, Q_2)) \in \delta^*$ with $f_1 \in F_1$ and $Q_2 \cap F_2 = \emptyset$. Using (5) we infer that $(\iota_1, tr, f_1) \in \delta_1^*$, implying that $tr \in \mathsf{Acc}(A_1)$. Next assume, for the sake of arriving at a contradiction, that there exists $tr^+ \in \mathsf{Acc}(A_2)$ with $tr \leqslant tr^+$. That is, there exists $f_2 \in F_2$ such that $(\iota_2, tr^+, f_2) \in \delta_2^*$. Using (6) we infer that $f_2 \in Q_2$, contradicting $Q_2 \cap F_2 = \emptyset$.

For "$\supseteq$", our assume that $tr \in \mathsf{Acc}(A_1)$ but there does not exists $tr^+ \in \mathsf{Acc}(A_2)$ with $tr \leqslant tr^+$. Thus there exists $f_1 \in F_1$ such that $(\iota_1, tr, f_1) \in \delta_1^*$, and with $Q_2 = \{q_2 \mid \exists tr^+ \geqslant tr : (\iota_2, tr^+, q_2) \in \delta_2^*\}$ we have $Q_2 \cap F_2 = \emptyset$. Using (9) we infer that $((\iota_1, \{\iota_2\}), tr, (f_1, Q_2)) \in \delta^*$, which since $(f_1, Q_2) \in F$ shows that $tr \in \mathsf{Acc}(A_1 \setminus^{\leq} A_2)$.

This finishes the proof of the second line of Lemma 6.18. It is an easy exercise to verify that the above can be converted into a proof of the first line of the lemma, by replacing all occurrences of "$\leqslant$" (and "$\geqslant$") by "$=$". □

Lemma 6.23 Let $A = (\mathcal{Q}, \iota, F, \delta)$ and H_0 be given, and assume that tr is saturated. If $q \in \mathcal{Q}$ is such that $(\iota, tr, q) \in \delta^*$, then $Dest(H_0, tr) \subseteq \mathsf{H}^A_{H_0}(q)$.

In particular, if tr is feasible from H_0 then $\mathsf{H}^A_{H_0}(q) \neq \emptyset$.

Proof. The last part follows obviously from the first part. For the first part, let $tr = a_1 \dots a_n$ ($n \geq 0$). There exists $q_1 \dots q_n$ and $q'_1 \dots q'_n$ such that with $q_0 = \iota$ we have for all $i \in \{1 \dots n\}$

$$(q_{i-1}, \bullet, q'_i) \in \delta^* \text{ and } (q'_i, a_i, q_i) \in \delta$$

and additionally $(q_n, \bullet, q) \in \delta^*$. We shall establish the following result:

$$\forall i \in \{0 \dots n\} : Dest(H_0, a_1 \dots a_i) \subseteq \mathsf{H}^A_{H_0}(q_i) \text{ unless } i \geq 1 \text{ with } a_i \text{ of the form } \mathsf{put}(\_). \tag{11}$$

We prove (11) by induction in i. For $i = 0$, the claim is that $H_0 \subseteq \mathsf{H}^A_{H_0}(\iota)$ which follows immediately since $\mathsf{H}^A_{H_0}$ is valid.

The case $i > 0$ with a_i of the form $\mathsf{get}(\_)$. From tr being saturated we infer that $i - 1 > 0$ with a_{i-1} of the form $\mathsf{put}(\_)$, and that it does not hold that $i - 2 > 0$ with a_{i-2} of the form $\mathsf{put}(\_)$. Therefore, we can inductively assume that

$$Dest(H_0, a_1 \dots a_{i-2}) \subseteq \mathsf{H}^A_{H_0}(q_{i-2})$$

and since $(q_{i-2}, \mathsf{put}(\_)\ \mathsf{get}(\_), q_i) \in \delta^*$ we infer from $\mathsf{H}^A_{H_0}$ being valid that

$$\mathsf{H}^A_{H_0}(q_{i-2}) \subseteq \mathsf{H}^A_{H_0}(q_i).$$

This implies (employing Lemma 3.14) the desired relation

$$Dest(H_0, a_1 \dots a_i) = Dest(Dest(Dest(H_0, a_1 \dots a_{i-2}), \mathsf{put}(\_)), \mathsf{get}(\_)) = Dest(H_0, a_1 \dots a_{i-2}) \subseteq \mathsf{H}^A_{H_0}(q_i).$$

The case $i > 0$ with a_i not of the form $\mathsf{get}(\_)$. It cannot be the case (again since tr is saturated) that $i - 1 > 0$ with a_{i-1} of the form $\mathsf{put}(\_)$. Therefore, we can inductively assume that

$$Dest(H_0, a_1 \dots a_{i-1}) \subseteq \mathsf{H}^A_{H_0}(q_{i-1}).$$

Moreover, we infer from $\mathsf{H}^A_{H_0}$ being valid that

$$\mathsf{H}^A_{H_0}(q_{i-1}) \subseteq \mathsf{H}^A_{H_0}(q'_i)$$

and since a_i is positional also that

$$Dest(\mathsf{H}^A_{H_0}(q'_i), a_i) \subseteq \mathsf{H}^A_{H_0}(q_i).$$

The desired relation now follows (employing Lemmas 3.14 and 3.13) from the calculation

$Dest(H_0, a_1 \ldots a_i) = Dest(Dest(H_0, a_1 \ldots a_{i-1}), a_i) \subseteq Dest(\mathsf{H}^A_{H_0}(q_{i-1}), a_i) \subseteq Dest(\mathsf{H}^A_{H_0}(q'_i), a_i)$
$\subseteq \mathsf{H}^A_{H_0}(q_i)$.

We have thus established (11). This proves the lemma: since it is not possible (as tr is saturated) that $n \geq 1$ with a_n of the form $\mathsf{put}(\_)$, (11) tells us that

$$Dest(H_0, a_1 \ldots a_n) \subseteq \mathsf{H}^A_{H_0}(q_n) \subseteq \mathsf{H}^A_{H_0}(q)$$

where the last inequality follows from $(q_n, \bullet, q) \in \delta^*$ and $\mathsf{H}^A_{H_0}$ being valid. □

Lemma 6.24 Let $A = (\mathcal{Q}, \iota, F, \delta)$ and H_0 be given. If $g \in \mathsf{H}^A_{H_0}(q)$ with $q \in \mathcal{Q}$, then there exists tr feasible from H_0 with $(\iota, tr, q) \in \delta^*$ and $g \in Dest(H_0, tr)$.

Proof. The result will follow if we can establish that during execution of the algorithm in Fig 8 it always holds that

if $g \in \mathsf{H}(q)$ then $\exists tr$ feasible from H_0 with $(\iota, tr, q) \in \delta^*$ and $g \in Dest(H_0, tr)$.

We prove this result by induction in the iteration. Initially, it must be the case that $q = \iota$ and $g \in H_0$, so clearly $tr = \bullet$ does the job.

For the inductive step, assume that g is added to $\mathsf{H}(q)$ during an iteration. There are three cases:

$(q', \epsilon, q) \in \delta$ with $g \in \mathsf{H}(q')$. Inductively, there exists tr feasible from H_0 with $(\iota, tr, q') \in \delta^*$ and $g \in Dest(H_0, tr)$. But then also $(\iota, tr, q) \in \delta^*$.

$(q', \mathsf{put}(\sigma)\,\mathsf{get}(\sigma'), q) \in \delta^*$ with $g \in \mathsf{H}(q')$. Inductively, there exists tr' feasible from H_0 with $(\iota, tr', q') \in \delta^*$ and $g \in Dest(H_0, tr')$. Let $tr = tr' \diamond \mathsf{put}(\sigma)\,\mathsf{get}(\sigma')$, then tr is feasible from H_0 with $(\iota, tr, q) \in \delta^*$ and $g \in Dest(H_0, tr)$.

$(q', a, q) \in \delta$ with a positional and $g \in Dest(\mathsf{H}(q'), a)$. By Lemma 3.15, there exists $g' \in \mathsf{H}(q')$ such that $g \in Dest(g'^{\uparrow}, a)$. By applying the induction hypothesis on $g' \in \mathsf{H}(q')$, we infer that there exists tr' feasible from H_0 with $(\iota, tr', q') \in \delta^*$ and with $g' \in Dest(H_0, tr')$, implying $g'^{\uparrow} \subseteq Dest(H_0, tr')$. Let $tr = tr' \diamond a$, then $(\iota, tr, q) \in \delta^*$ and (using Lemmas 3.13 and 3.14)

$$g \in Dest(g'^{\uparrow}, a) \subseteq Dest(Dest(H_0, tr'), a) = Dest(H_0, tr)$$

also showing that tr is feasible from H_0. □